DESPLIEGUE DE UNA RED DE SENSORES BASADA EN CHIPS ESP-8266

ALDA MARTÍN MUÑOZ

MÁSTER EN INTERNET DE LAS COSAS. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Internet de las Cosas

Convocatoria: Enero/Febrero de 2020
Calificación: 8 (Notable)

Directores:

Alberto Antonio del Barrio
Guillermo Botella

Resumen en castellano

Hoy en día vivimos en una sociedad que cada día se rodea de más tecnología y aparatos electrónicos que facilitan el día a día. La gran parte de los dispositivos que se usan en la vida cotidiana se conectan a internet, lo que permite que los usuarios se puedan comunicar de manera inmediata desde casi cualquier parte del mundo y por diferentes vías tanto entre personas como personas con dispositivos o dispositivos con dispositivos. El deseo de controlar automáticamente y de forma remota de muchos de los sistemas que se utilizan de forma regular ha dado lugar a la aparición del Internet de las cosas o IoT (del inglés, *Internet of Things*). En el IoT todos los dispositivos están conectados entre ellos y son capaces de intercambiar información entre ellos con el objetivo de realizar una tarea concreta.

El desarrollo del IoT ha generado que se implementen nuevos dispositivos y tecnologías que permitan la optimización de su funcionamiento. También, el creciente interés de algunos usuarios por el conocimiento del funcionamiento de estos sistemas ha generado aparatos y programas al alcance de personas no profesionales.

Este trabajo presenta el panorama actual de los chips de bajo coste al alcance de una gran parte de la población y analiza distintos tipos de redes de IoT implementadas con dispositivos ESP8266. Se ha realizado una comparativa entre dos topologías, *mesh* y estrella, combinadas con con dos protocolos cada una: MQTT y TCP. Una vez analizados los resultados de la misma, se ha desarrollado un ejemplo de un caso real de aplicación con una de las redes analizadas. En concreto, se trata de un sistema domótico que detecte posibles anomalías en el comportamiento del inquilino con el objetivo de proporcionar ayuda si es necesario.

Palabras clave

Internet de las cosas, IoT, topologías, MQTT, TCP, ESP8266, Arduino, WiFi, sistema domótico.

Abstract

Nowadays we live in a society where we are surrounded by technology and electronic devices that make day-to-day life easier. Most of the devices that are used in daily life are connected to the Internet, which allows users to communicate between themselves and also enables devices to communicate between them from almost anywhere in the world. The desire to automatically and remotely control many of the systems used frequently has led to the emergence of the Internet of Things (IoT). In the IoT all devices are connected to each other and are able to exchange information between them in order to perform a specific task.

The development of the IoT has generated the implementation of new devices and technologies to optimize its operation. Also, the growing interest of some users for the knowledge of the operation of these systems has generated devices and programs within the reach of non-professionals.

This paper presents the current state of low cost chips within the reach of a large part of the population and analyzes different types of IoT networks implemented with ESP8266 devices. A comparison has been made between mesh and star topologies, which are combined with MQTT and TCP protocols, respectively. In addition, an example of a real application case of one of the analyzed networks has been deployed. Specifically, it is a home automation system that detects possible anomalies in the behavior of the tenant with the aim of providing assistance if necessary.

Keywords

Internet of things, IoT, topologies, MQTT, TCP, ESP8266, Arduino, WiFi, home automation system.

Índice general

Lista de Figuras

Lista de Tablas

Agradecimientos

Gracias a mis padres y a mi hermana por apoyarme y ayudarme en todo momento.

A mis amigos por acompañarme en los buenos momentos de la carrera y del máster y en algunos no tan buenos.

A mis tutores por guiarme en el desarrollo del proyecto.

Capítulo 1

Introducción y objetivos

1.1. Motivación

Hoy en día vivimos en un mundo en el que el desarrollo y le presencia de la tecnología en nuestras vidas aumenta casi por momentos. El objetivo de querer comunicarse entre distintas partes del planeta a través de dispositivos tecnológicos en tiempo real ha dado como resultado un mundo hiperconectado en el que prácticamente en cualquier lugar se tiene acceso a la mayor red de datos, internet.

Desde los primeros ordenadores que se conectaron a internet hasta hoy, la evolución de la tecnología ha sido exponencial. Actualmente muchos de los aparatos que se utilizan en la vida diaria tienen integrados chips programables que pueden conectarse a internet con el objetivo de facilitar aún más el uso de los mismos. A esta expansión de las conexiones a internet en objetos de uso cotiniano más allá de los ordenadores se la nombró como Internet de las Cosas o IoT (*Internet of things* en inglés) por primera vez en 1999[1].

Hoy en día el Internet de las cosas está muy extendido en todos los niveles, desde la industria o la agricultura a la vida cotidiana. A raíz de esto han surgido multitud de tecnologías y dispositivos que tienen como objetivo que los usuarios del IoT comprendan su funcionamiento. Estos componentes electrónicos con objetivos didácticos son muy variados y se adaptan a las necesidades de muchos proyectos *caseros*, como pueden ser las placas

[1] http://www.bcendon.com/el-origen-del-iot/

de desarrollo Arduino o RaspberryPi. La existencia de estos dispositivos y el interés de los usuarios ha generado unas redes de comunicación entre ellos que han derivado en algunos casos a la existencia de grupos de desarrolladores de nuevas funcionalidades para los chips, así como la generación de nuevas ideas de proyectos.

Este trabajo busca analizar algunas estas nuevas tecnologías al alcance de la mayoría de la población en lo relativo a redes de Internet de las cosas. Este tipo de dispositivos son accesibles para los usuarios sin muchos conocimientos técnicos debido a la amplia documentación existente sobre los mismos y su bajo coste. La expansión del IoT ha provocado también que estos chips incluyan conexiones a internet y librerías que permiten implementar redes con estructuras propias, por ello se analizarán algunas de estas funcionalidades y se comprobará el funcionamiento de estas redes en casos reales que permitan obtener datos sobre el comportamiento de las mismas.

Aunque este tipo de plataformas tengan como objetivo aprender sobre las tecnologías su uso no se limita a pequeños proyectos. Existen grandes despliegues con ellas que son funcionales.

1.2. Objetivos

Este proyecto tiene como objetivo principal analizar el funcionamiento de distintos tipos de redes formadas por dispositivos ESP8266 según la tecnología con la que se implementan. Para ello, se han comparado dos protocolos y dos topologías que se han combinado entre sí, dando lugar a cuatro escenarios de pruebas distintos. De este objetivo pueden derivarse algunos más específicos:

- Estudiar y documentar la situación actual de los dispositivos utilizados para IoT de bajo coste y sencillos de utilizar.
- Estudiar las distintas tecnologías que pueden utilizarse para implementar redes IoT con este tipo de dispositivos.

- Estudiar distintas topologías y protocolos utilizados en IoT a analizar.
- Realizar el análisis y comparativa de funcionamiento entre las redes seleccionadas.

Una vez analizados y comparados los distintos tipos de redes, se implementará un prototipo de un sistema domótico real que haga uso de una de las tecnologías analizadas. El objetivo principal de la implementación de este prototipo es comprobar que la red es útil para la aplicación, y de él se pueden especificar los siguientes objetivos específicos:

- Diseñar el sistema completo: especificaciones, funcionalidad, componentes y tecnologías a utilizar.
- Implementar la red completa, todos los nodos y componentes que la forman.
- Realizar prueba final y comprobar que se cumplen las especificaciones de la misma.

1.3. Organización del documento

Este documento está organizado en distintos capítulos, con la siguiente estructura:

- **Capítulo 1: Introducción.** En este capítulo se realiza una presentación del proyecto, donde se incluye la motivación, los objetivos y la estructura del mismo.
- **Capítulo 2: Estado del arte.** En esta sección se detalla el estado actual de los dispositivos de bajo coste para redes IoT que además sean sencillos de programar, presentando los más utilizados.
- **Capítulo 3: Entorno de desarrollo.** En él se enumeran los materiales físicos y plataformas y herramientas informáticas utilizadas para la realización del proyecto.
- **Capítulo 4: Marco teórico.** En este capítulo se desarrolla el marco teórico de las tecnologías utilizadas en el proyecto.

- **Capítulo 5: Análisis y resultados obtenidos.** En esta sección se analizan las distintas redes y comentan los resultados obtenidos.

- **Capítulo 6: Escenario real de aplicación.** Con el objetivo de darle un enfoque realista al proyecto, se ha implementado una aplicación real para un escenario concreto de una de las redes analizada que se detalla en este Capítulo 6.

- **Capítulo 7: Conclusiones.** Finalmente, se resulme el trabajo realizado y se comentan las implicaciones del mismo.

Capítulo 2

Estado del arte

La evolución de la tecnología y del internet de las cosas es constante. Cada vez hay más dispositivos que se conectan entre ellos sin necesidad de que un ser humano intervenga para captar información del entorno y poder actuar en función de ella. Este tipo de interacción se conoce como máquina a máquina o M2M (*Machine to Machine* en inglés) y cada vez está más presente en la vida cotidiana.

Desde los primeros dispositivos M2M, los sistemas electrónicos y las conexiones entre ellos han cambiado mucho. El IoT ha ido evolucionando hacia grandes sistemas distribuidos. Las placas que controlan los sensores y los actuadores cada vez tienen más prestaciones y son capaces de procesar algunos datos antes de mandarlos o utilizarlos para lo que se desee. Las redes IoT suelen tener varias placas para controlar los distintos sensores que la componen y últimamente se están utilizando para aligerar los datos que llegan a mandarse a la red, preprocesando los datos captados. La red de comunicación que conecta los sensores o actuadores entre ellos puede conducir o no los datos a internet, depende del objetivo con el que se observe el entorno.

Por otra parte, el desarrollo de la tecnología ha hecho que los dispositivos electrónicos hayan disminuido su precio y el coste de fabricación. Esto ha conseguido que montar redes IoT sea cada vez más barato y por tanto, que se estén implementando con mucha frecuencia. Además, el creciente interés por la tecnología y la programación por parte de gente no profesional ha provocado que se desarrollen sistemas electrónicos preparados para montar-

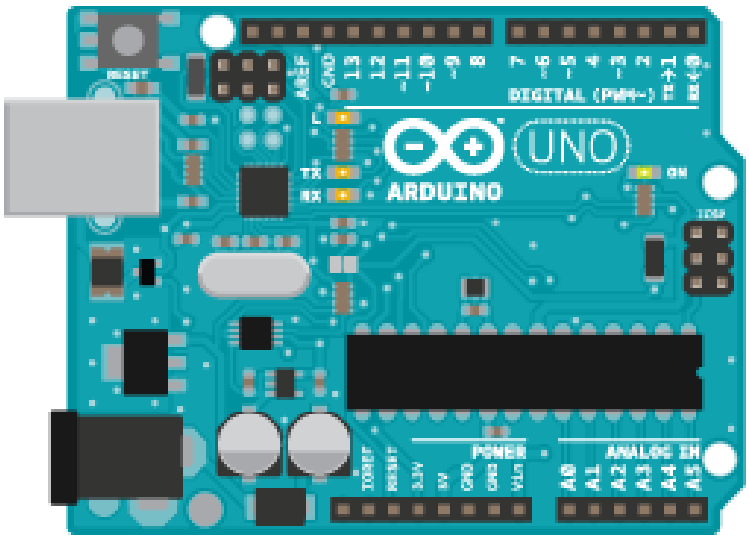

Figura 2.1: *Placa Arduino UNO.*

se y programarse de forma sencilla e intuitiva sin que el precio suba en exceso. Internet también ha facilitado que se generen comunidades de personas que intercambian ideas y conocimientos sobre electrónica a través de foros o redes sociales y consiguen hacer grandes proyectos.

Dentro del mundo profesional y educativo del IoT y la electrónica hay distintos tipos de *hardware* y *software* que se utilizan según el desarrollo de la tecnología de lo que se va a hacer y los objetivos que se tengan.

Una de las placas que más se utiliza tanto en proyectos personales como en el mundo profesional es Arduino, ya que es una plataforma abierta (*open source*) que facilita la utilización y programación de la electrónica. Arduino es un proyecto que dispone de dos partes, una que se centra en el *hardware* de las placas controladoras y los distintos sensores y actuadores que se pueden conectar y otra que se centra en la programación de este hardware utilizando un entorno y un lenguaje propio. La más utilizada es la placa Arduino UNO que podemos ver en la Figura 2.1.

Arduino dispone de un gran número de sensores y actuadores que se pueden conectar directamente a su placa de forma sencilla, lo que facilita mucho el proceso de desarrollo del

prototipo de un proyecto, o incluso del proyecto en sí mismo. Dentro de estos periféricos se pueden encontrar todo tipo de sensores y actuadores: temperatura, humedad, luminosidad, ruido, motores, zumbadores, relés, LEDs... Además también hay accesorios para las placas que añaden funcionalidades al Arduino básico, como dispositivos WiFi o Bluetooth, controladores de motores o antenas.

Recientemente, el proyecto Arduino se ha actualizado y ahora mismo ofrece múltiples placas controladoras con distintas características que se adaptan al nivel de conocimientos del usuario. Cabe destacar que todas utilizan controladores ATmega, aunque con distintas especificaciones. La placa mencionada anteriormente, Arduino UNO, se incluye dentro de las recomendadas para principiantes igual que las nuevas Arduino Leonardo, Arduino Esplora o las pequeñas Arduino Nano y Arduino micro. Todas excepto la llamada Esplora son placas con un controlador y distintos pines GPIO que permiten conectar distintos periféricos de forma sencilla. La placa Esplora está basada en la Leonardo e incluye algunos botones y sensores listos para utilizarse nada más enchufarla[1].

El número de usuarios de Arduino es cada vez mayor, por lo que aunque las placas que se han llamado *para principiantes* permitan hacer proyectos muy complejos, se han implementado nuevos dispositivos con más capacidad de procesamiento y más pines GPIO para conectar más periféricos. Entre ellos pueden encontrarse los más clásicos como Arduino Mega o Due, que tienen 54 pines GPIO digitales y 12 analógicos y las nuevas placas que vienen preparadas para el Internet de las cosas. Es importante destacar la cantidad de placas controladoras y módulos de accesorio que ha desarrolado Arduino para facilitar el uso de sus dispositivos en los proyectos IoT. Hay disponibles algunas con conexión a internet a través de WiFi, Ethernet, SigFox, GSM, Lora o LTE. Además ha incorporado en su web un editor de código online y una nube que permite controlar y visualizar los distintos dispositivos que se quieran añadir a nuestra red IoT[1]. Aunque estas placas facilitan mucho la realización de proyectos tienen un inconveniente principal, y es que el precio es alto para la cantidad de ellas que harían falta en una red real de IoT. Debido a su versatilidad, están destinadas más

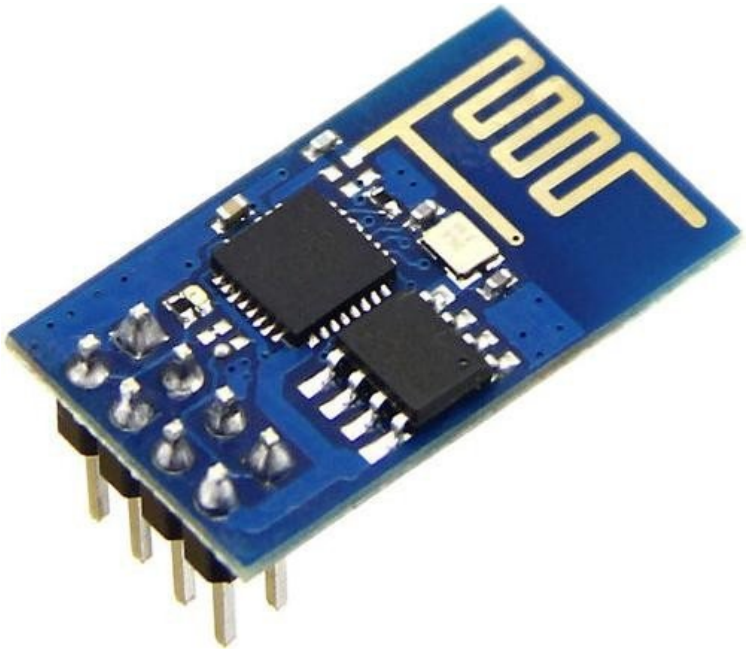

Figura 2.2: *Módulo WiFi ESP8266 de Espressif.*

a proyectos prototipo, educativos o aficionados.

Por otra parte, cabe destacar que uno de los módulos que más se ha utilizado durante años, antes de que se presentaran estos nuevos módulos IoT, ha sido el chip integrado ESP8266[7] (Figura 2.2), que permite que el controlador Arduino se conecte a Internet a través de WiFi y es compatible con el protocolo TCP/IP. Este dispositivo lo fabrica Espressif y aunque se utilice para conectarse a otro controlador, también se puede utilizar y programar por separado, ya que dispone de puertos GPIO donde se le pueden conectar sensores. Además su precio ronda los 3€[1], es barato y consume muy poco, lo que hace de él un chip muy recomendable para proyectos IoT.

El ESP8266 lleva una CPU de 32 bits y dispone de 8 pines digitales y uno analógico, además de soportar los protocolos de comunicación SPI, I2C y UART. El consumo del chip es bastante reducido, oscila entre los 0,5 microamperios cuando está en espera y 170 miliamperios al transmitir señales. Al ser un dispositivo pequeño y orientado a IoT, se busca

[1] http://es.aliexpress.com/wholesale?catId=0&initiative_id=AS_20200107100240&SearchText=nodemcu+esp8266

que consuma lo mínimo posible, por lo que tiene a disposición del usuario tres modos de trabajo[3]:

- Modo *active*: el que el dispositivo está a pleno rendimiento.
- Modo *sleep*: el dispositivo está a bajo rendimiento, pendiente sólo de los eventos que se hayan predefinido que puedan despertarle. Guarda todos los datos en la memoria, por lo que cuando despierta no es necesario reiniciar las conexiones.
- Modo *deep sleep*: el chip está encendido pero no está operativo. En este estado los datos que estuvieran en la memoria se pierden.

Cabe destacar que este chip se integra dentro de distintas placas de diferentes fabricantes. La más utilizada es la *NodeMCU*, que sigue la filosofía Arduino y viene lista para enchufarla y programarla, ya que dispone de módulo USB y un firmware que permite programarla en distintos lenguajes. Puede verse en la Figura 2.3.

Aunque Arduino tiene placas con distintas características, todas son controladores muy básicos que tienen limitaciones en software y sistema operativo. Por este motivo, hay situaciones en las que se necesita añadir a los proyectos algún controlador que tenga sistema operativo o más prestaciones en cuanto a velocidad o comunicaciones. Uno de los micro ordenadores más utilizados para ello es la Raspberry Pi, debido a las prestaciones que tiene y su tamaño reducido. Normalmente, este tipo de dispositivos se utilizan para procesar algunos datos recogidos por los sensores de la red, para controlarlos o conectar la red a internet. Igual que el proyecto Arduino, Raspberry Pi también tiene distintos modelos de placas *hardware* con distintas prestaciones que se adaptan a las necesidades del proyecto.

Todas las placas Raspberry Pi tienen conexión a internet, a través de WiFi y Ethernet y se las puede conectar distintos periféricos directamente, como sensores, actuadores o pantallas. Muchas veces se utilizan conjuntamente Raspberry Pi y Arduino, ya que hay muchas ocasiones en las que para controlar sensores distribuidos en el espacio es suficiente con placas Arduino, que después se conectan con la Raspberry Pi. Además, el precio de la

Figura 2.3: *Módulo NodeMCU.*

Raspberry Pi es superior al de Arduino, y normalmente las prestaciones de la Raspberry son muy superiores a lo que necesita el proyecto.

Por ejemplo, la Raspberry Pi 4 Model B, que puede verse en la Figura 2.4 y es la que se está comercializando ahora, tiene distintos modelos con memorias RAM diferentes para elegir de 1, 2 o 4 GB y después incorpora dos puertos micro HDMI que permiten dos monitores simultáneos, puerto Ethernet, WiFi, Bluetooth y puertos USB tanto 2.0 y como 3.0 por un precio de 35 dólares [2]. Los modelos anteriores son más sencillos y las principales diferencias que incorporan con este es que no permiten doble pantalla simultánea ni tienen USB 3.0. Sin embargo, una placa de Raspberry Pi que merece la pena mencionar es el modelo Zero W, que es mucho más pequeña que las convencionales y algo más simple en cuanto a prestaciones, pero siguiendo la misma filosofía. Como puede verse en la Figura 2.5

[2] https://www.raspberrypi.org/products/

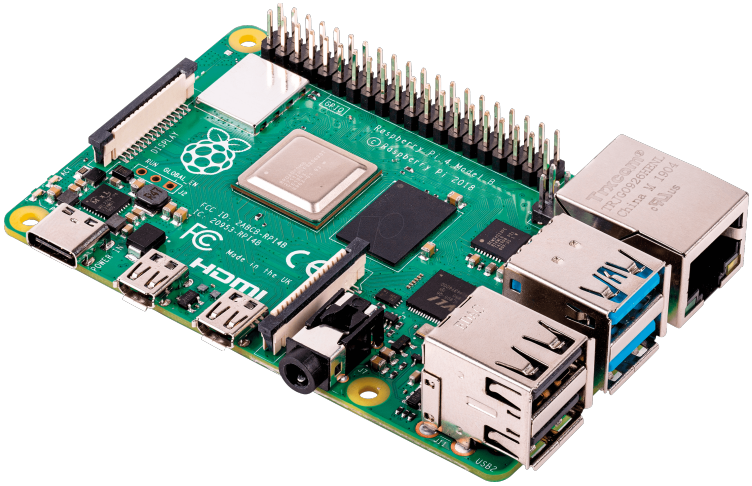

Figura 2.4: *Micro ordenador Raspberry Pi 4.*

Figura 2.5: *Micro ordenador Raspberry Pi Zero W.*

también incorpora puertos micro USB y micro HDMI, WiFi y Bluetooth[3].

Otra alternativa de código abierto a estos dispositivos son las placas que están basando su arquitectura en RISC-V, que tiene un conjunto de instrucciones totalmente gratuito y abierto. Este tipo de procesadores empezaron siendo desarrollados en la Universidad de Berkeley en 2010 [4], pero también se unieron como colaboradores distintas empresas y voluntarios.

Una de las empresas que ha basado sus chips en esta arquitectura ha sido Seeed. En

[3] https://www.raspberrypi.org/products/raspberry-pi-zero-w/
[4] https://riscv.org/risc-v-history/

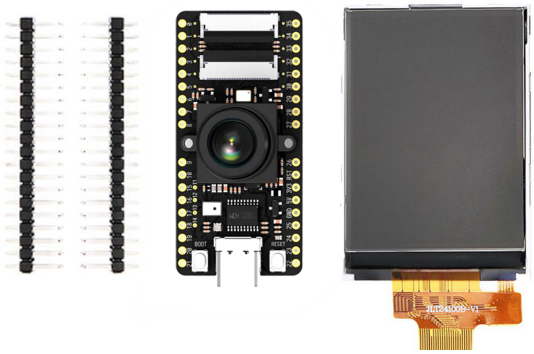

Figura 2.6: *Placa Sipeed Maix con cámara y pantalla LCD.*

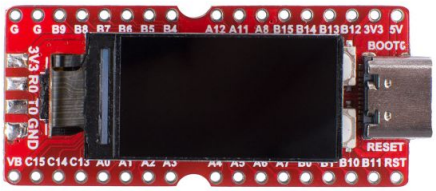

Figura 2.7: *Placa Sipeed Logan Nano.*

concreto, ha desarrollado las placas de desarrollo para IoT Sipeed MAIX, que pueden verse en la Figura 2.6 [5] que facilitan la incorporación de periféricos como pantallas LCD o cámaras además de los tradicionales GPIO para el resto de conexiones. La potencia de estos chips es bastante superior a los mencionados anteriormente en esta sección, ya que incorporan procesadores de 64 bits y doble núcleo. Esta marca también ha desarrollado otros chips más pequeños y baratos, que se asemejan más a los Arduinos o ESP8266 y son los Sipeed Logan Nano [6], que como puede verse en la Figura 2.7 incorporan una pantalla LCD más pequeña. El precio de estos chips Sipeed Logan Nano ronda los 6€[7], son competencia de los demás chips vistos en este apartado.

[5] https://www.seeedstudio.com/Sipeed-MAix-BiT-for-RISC-V-AI-IoT-1-p-2873.html

[6] https://www.seeedstudio.com/Sipeed-Longan-Nano-RISC-V-GD32VF103CBT6-Development-Board-p-4205.html

[7] http://es.aliexpress.com/item/4000482853487.html?spm=a2g0o.productlist.0.0.29fd5a6aNxr4Mu&algo_pvid=407982f8-01ee-48b5-86e5-de366f56750d&algo_expid=407982f8-01ee-48b5-86e5-de366f56750d-0&btsid=38f7fbbc-2779-4beb-966b-53445d81e32a&ws_ab_test=searchweb0_0,searchweb201602_6,searchweb201603_53

Capítulo 3

Entorno de desarrollo

Para la realización de este proyecto se han utilizado una serie de materiales y herramientas que han permitido el análisis de las distintas topologías y protocolos de comunicación desarrollada en el Capítulo 5 y la implementación de la aplicación final del Capítulo 6.

3.1. Materiales

Para la realización del análisis de los distintos tipos de redes se han utilizado los materiales físicos que se indican a continuación.

- 8 placas NodeMCU ESP8266 de Espressif.
- 9 cables USB y transformadores de corriente para alimentar las placas.
- 1 powerbank.
- 1 ordenador sobremesa con procesado Intel i7, 24 GB de ram, disco duro SSD de 264 GB y 3 puertos USB 3.0 y 7 puertos 2.0.
- 1 Raspberry Pi 3 Model B.
- Multímetro digital.

Para la implementación de la aplicación final en la que se aplica una de las redes analizada se han utilizado los siguientes materiales:

- 1 ordenador sobremesa con procesado Intel i7, 24 GB de ram, disco duro SSD de 264 GB y 3 puertos USB 3.0 y 7 puertos 2.0.
- 4 placas NodeMCU ESP8266 de Espressif.
- 1 placa Heltec WiFi Kit 8 con LCD basada en ESP8266.
- 1 Raspberry Pi 3 Model B.
- 6 cables USB y transformadores de corriente para alimentar las placas.
- 1 powerbank.
- 1 sensor de llama para Arduino KY-026.
- 1 sensor de luminosidad LDR KY-018.
- 1 sensor infrarrojo KY-033.
- 1 zumbador.
- 2 pulsadores.
- Cables de un componente.
- 2 placas de inserción.

Para los dos casos ha sido necesario también tener acceso a una red WiFi a la que conectar los dispositivos de la red. También la conexión a internet para la instalación del software y bibliotecas necesarias.

3.2. Herramientas

Como herramientas de desarrollo tanto para el análisis como para la implementación del escenario de aplicación se han utilizado:

- Framework de RTOS de Espressif para ESP8266. Con este framework se realizaron algunas pruebas pero finalmente se descartó su utilización.
- Framework de Arduino para ESP8266 y diferentes librerías del mismo. Framerwork que finalmente se ha utilizado en todo el trabajo.
- Visual Studio Code con Platformio para programar los ESP8266.
- Hojas de cálculo de Google para procesar los datos obtenidos.
- GitHub.
- NodeRED en la implementación del sistema real de aplicación.

Capítulo 4

Marco teórico

En esta sección se analizan distintas topologías y protocolos de comunicación entre los dispositivos ESP con el objetivo de ver los aspectos más y menos favorables de cada una de las combinaciones.

4.1. Protocolos de red

En este trabajo se va a trabajar con dos protocolos de red: TCP y MQTT. TCP es un protocolo de la capa de transporte y MQTT de la capa de aplicación que se implementa sobre TCP u otros protocolos de transporte.

4.1.1. Protocolo TCP

El protocolo TCP o Protocolo de Control de Transmisión es uno de los protocolos básicos y de los más utilizados, ya que proporciona una entrega continua fiable de datos entre los sistemas que estén conectados.

En una comunicación de dos dispositivos a través del protocolo TCP se desempeñan dos papeles diferentes. El emisor del mensaje hará de cliente y el dispositivo que tiene que recibirlo será el servidor, por ello este protocolo es del tipo cliente/servidor. El protocolo TCP se caracteriza por el envío de acuses de recibo cada vez que el cliente o el servidor reciben un paquete. Esta confirmación de recepción de datos se conoce como ACK, abreviatura de inglés *acknowledgement* que significa acuse de recibo o asentimiento.

Offsets	Octeto	0								1								2								3							
Octeto	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Puerto de origen																Puerto de destino															
4	32	Número de secuencia																															
8	64	Número de acuse de recibo (si ACK es establecido)																															
12	96	Longitud de Cabecera				Reservado				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Tamaño de Ventana														
16	128	Suma de verificación																Puntero urgente (si URG es establecido)															
20 ...	160 ...	Opciones (Si la Longitud de Cabecera > 5, relleno al final con "0" bytes si es necesario) ...																															

Figura 4.1: *Esquema del formato de la cabecera de los mensajes TCP*[16].

La estructura de los mensajes en el protocolo TCP es fija y siempre tiene el mismo formato. Según el tipo de mensaje del que se trate o los datos que deba transportar, se cambian los valores de los campos. Entre otros, el datagrama incluye los puertos de origen y destino, el número de secuencia o el bit ACK de confirmación. Esta estructura de mensaje puede verse en la Figura 4.1.

Para crear esta conexión son necesarias tres etapas que se conocen como *negociación en tres pasos*[16]. Estos tres pasos pueden verse en el diagrama de la Figura 4.2 y consisten en los siguientes:

- El cliente realiza una llamada al servidor para abrir el puerto en el que se va a realizar la conexión a través de un paquete SYN.
- Al recibir este paquete, el servidor comprueba si el puerto está disponible. En caso de que lo esté le devuelve al cliente la petición SYN añadiéndole un ACK de confirmación. Si el puerto no estuviera disponible, el servidor envía un paquete con el bit RST activado, que indica el rechazo de la conexión.
- Cuando el cliente reciba la respuesta del servidor, le confirma a través de un ACK que le ha llegado.

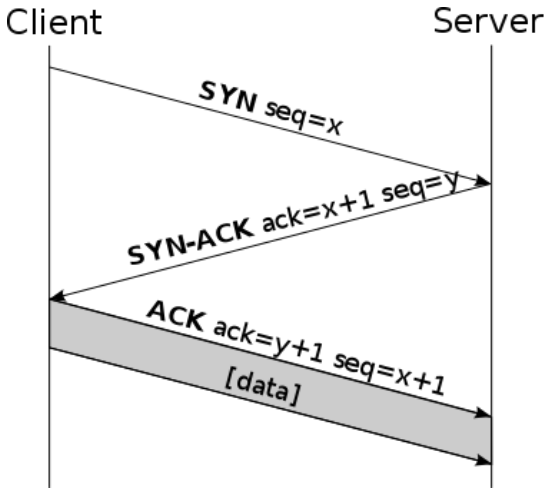

Figura 4.2: *Esquema del proceso de creación de conexión.*

Una vez realizados estos tres pasos, se procede a la transferencia de datos. Con el objetivo de que el traspaso de datos se realice de forma fiable y robusta, el protocolo dispone de distintas herramientas. Entre ellas se encuentran números de secuencia para poder ordenar los paquetes recibidos y detectar posibles anomalías, o *checksums* para detectar errores. También hay temporizadores que controlan las pérdidas o retrasos y ventanas deslizantes para el control de flujo en el envío de datos. La ventana deslizante consiste en que el paquete que envía el emisor incluye un campo en el que se especifica la cantidad de datos que puede almacenar el receptor antes de enviar el ACK de confirmación de recepción de datos. Lo que se busca con esto es aligerar el intercambio de paquetes entre el cliente y el servidor.

Al finalizar el envío de datos, la conexión TCP establecida entre el cliente y el servidor debe cerrarse y para ello se realiza una *negociación en cuatro pasos*. Para ello, el dispositivo que dé por finalizada la conexión debe enviar un paquete del tipo FIN y automáticamente ignora los mensajes que reciba a partir de ese momento. Cuando el otro dispositivo reciba el indicador FIN, le contestará con un ACK que lo ha recibido. Después de enviar el ACK, este mismo dispositivo debe cerrar también la conexión por su lado, por lo que es necesario que envíe un paquete FIN también y que la otra máquina le conteste con su ACK. Puede verse en la Figura 4.3 un esquema de este proceso de finalización de la conexión.

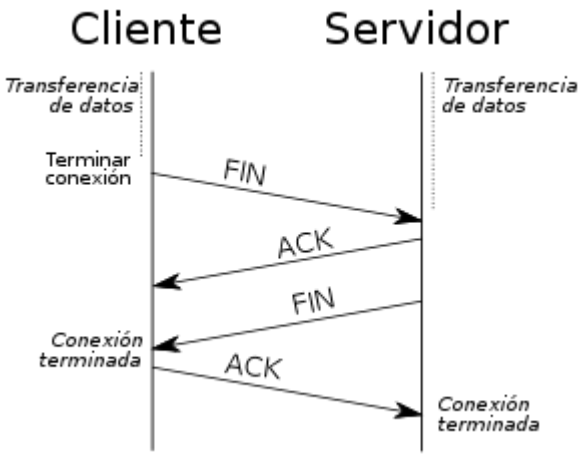

Figura 4.3: *Esquema del proceso de finalización de conexión TCP.*

4.1.2. Protocolo MQTT

Otro protocolo que se puede utilizar para la comunicación de dispositivos, y que además se utiliza mucho en IoT es MQTT. El protocolo, cuyas siglas significan *Message Queing Telemetry Transport*, se basa en mensajería asíncrona de publicar-suscribir y funciona sobre TCP/IP.

MQTT define dos tipos de entidades en la red[17]: un *broker* de mensajería y varios clientes que se conectan a él. Un *broker* es un servidor que recibe todos los mensajes de los clientes e inmediatamente los redirige a otros destinos que considere relevantes. Como cliente, se puede conectar cualquier elemento que pueda interactuar con el *broker* para enviar y/o recibir mensajes, como ordenadores o *smartphones*. El funcionamiento de MQTT hace que los mensajes puedan enviarse a través de un tema o canal de mensajería específico al que los clientes que quieran recibirlos deban suscribirse. Para que el envío de un mensaje también sea a un tema concreto, el cliente que lo mande debe indicar el tema además de el cuerpo del mensaje. El *broker* será el encargarlo de reenviar el mensaje a todos los clientes que estén suscritos a ese tema. La conexión entre los clientes y el *broker* puede ser TCP/IP simple o una TLS cifrada en caso de que los mensajes sean sensibles.

Un esquema de este tipo de conexión puede verse en la Figura 4.4. En este caso un sensor

manda un mensaje de temperatura con el topic *temp* y el broker se encarga de reenviarlo a todos los clientes que estén suscritos a este mismo topic.

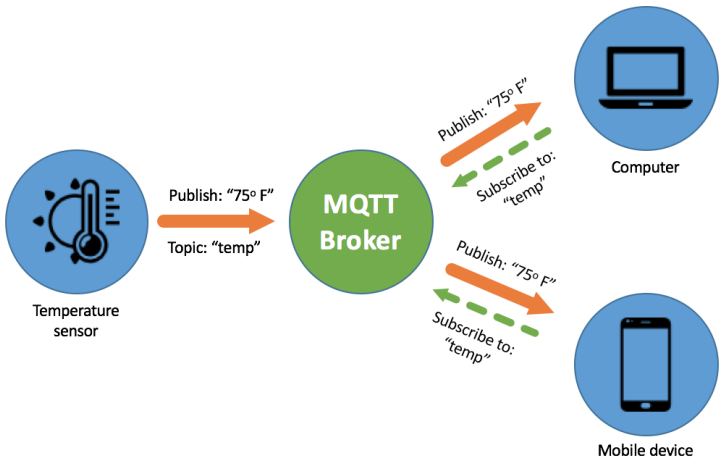

Figura 4.4: *Esquema básico de una red con protocolo MQTT*[10].

MQTT es un protocolo que se está utilizando mucho en IoT debido a su sencillez, por lo que se han desarrollado librerías para distintas plataformas que facilitan la implementación de sistemas con este protocolo. Una de las librerías más utilizadas es la librería *Mosquitto*, desarrollada por *Eclipse*, que además de para ordenadores, incluye soporte para clientes con C y C++. Tiene dos funciones básicas que permiten publicar mensajes y suscribirse a los *topics* deseados.

Para algunas plataformas como Arduino, hay librerías de cliente MQTT propias más sencillas, como la *PubSubClient* o *PainlessMesh*, que pueden tanto enviar mensajes como recibirlos.

4.2. Topologías

La topología de una red es la forma en la que están conectados los dispositivos que se encuentran dentro de ella. Dentro de la red, se conoce como nodo al punto de la red donde coinciden varias conexiones, que puede ser un *router* o *switch* o un dispositivo final como un ordenador, un teléfono o un sistema empotrado.

La conexión entre los nodos puede ser tanto alámbrica como inalámbrica, por lo que según el tipo de red que se quiera implementar será más conveniente usar una u otra topología. En el desarrollo de este trabajo se van a implementar redes inalámbricas y en concreto se harán pruebas con dos topologías: topología en malla y topología en estrella.

4.2.1. Topología *mesh*

El desarrollo del internet de las cosas durante los últimos años ha generado nuevas necesidades de comunicación entre los nodos de comunicación, ya que éstos están realizando tareas para las que las redes convencionales no estaban pensadas. Estas redes, se basan en que los dispositivos que quieran conectarse a internet lo hagan a través de un punto de acceso o *router*, que es el que se encarga de gestionar la conexión con el exterior, como se aprecia en la Figura 4.5. Actualmente, un *router* convencional permite tener conectados simultáneamente 32 dispositivos [15], una cifra que en algunas ocasiones no es suficiente para dar servicio a todos los sensores y actuadores de una red IoT. Este tipo de redes necesitan ser capaces de gestionar un número muy alto de nodos conectados simultáneamente, que normalmente se encuentran distribuidos por grandes extensiones de terreno y que necesitan acceso a Internet.

Para este problema, en este momento hay dos soluciones posibles: o bien utilizar un *router* de alta capacidad o, utilizar redes con topología *mesh*. Esta topología está implementada para que hasta 87 nodos de una red [15] puedan conectarse a Internet sin necesidad de que todos ellos estén conectados al punto de acceso. Los nodos de esta red se conectan entre ellos para intercambiar los paquetes de información dando los mínimos saltos posibles entre el dispositivo en cuestión y el *router*, como puede verse en la Figura 4.6. Podría decirse que los nodos de la red son pequeños *routers* que pueden cambiar el papel que desempeñan según las necesidades de cada momento.

El desarrollo de este apartado se centrará en el análisis del protocolo ESP-MESH implementado por Espressif [14], que permite que muchos nodos se conecten a una única red

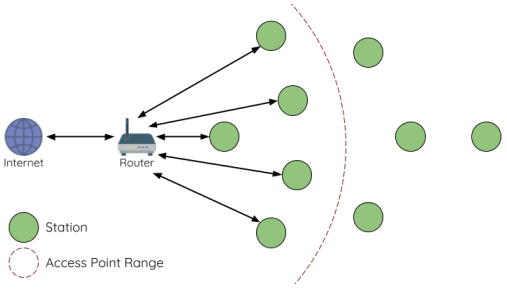

Figura 4.5: *Red WiFi tradicional.*

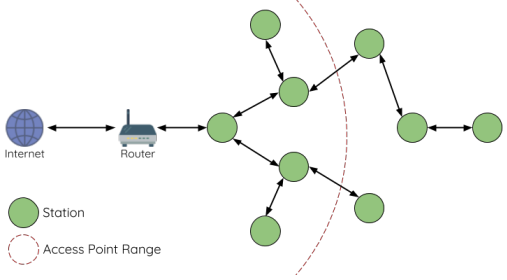

Figura 4.6: *Red Mesh.*

WLAN automáticamente, funcionando de manera autónoma. Para ello, es necesario que los nodos puedan tomar distintos papeles dentro de la misma red, según los nodos que haya conectados en ese instante. Esta topología sigue la estructura que se ve en la Figura 4.7, en la que hay distintos tipos de nodos.

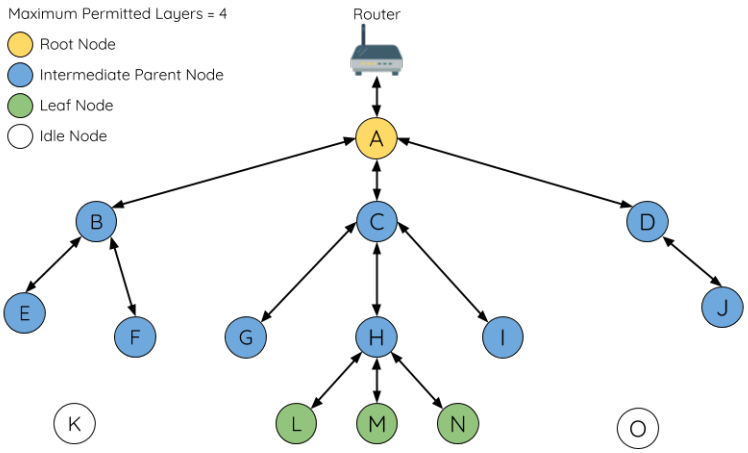

Figura 4.7: *Tipos de nodos de la red Mesh.*

- **Nodo raíz (*Root Node*)**: el nodo superior de la red en forma de árbol.
- **Nodo hoja (*Leaf Node*)**: un nodo sin hijos.
- **Nodo padre intermedio (*Intermediate Parent Node*)**: nodo con al menos un hijo.

- **Nodo en espera (*Idle Node*)**: nodo que aún no ha entrado a formar parte de la red.

Cada uno de los nodos de la red es capaz de formar nuevas conexiones hacia abajo dentro del árbol. Para que los nodos que aún no se han conectado a la red lo hagan, es necesario un sistema de avisos de presencia que permita a los nodos ver y ser vistos. En estos avisos, los nodos envían información sobre el papel que están desempeñando, la capa en la que se encuentran, los saltos necesarios para llegar al *router*, los hijos que tiene y el número máximo de hijos que tiene permitidos tener. Para que un nuevo nodo se conecte a la red y pueda elegir cuál será su nodo padre, tiene que escuchar a todos los nodos de la red que estén a su alcance y seleccionar el que más le convenga. Para ello, el nodo compara la capa en la que se encuentran los posibles nodos padres, el número de saltos al *router* y el número de hijos que ya tienen cada uno. Finalmente seleccionará el nodo padre que más cerca se encuentre del punto de acceso a le red, incluso aunque sea el nodo raíz y si todos están en la misma capa, se conectará con el nodo que menos hijos tenga.

Por ejemplo, en la Figura 4.8 podemos ver dos casos en los que un nodo G que aún no se ha conectado a la red, quiere hacerlo seleccionando un nodo padre. En el primero, el nodo G recibe avisos de los nodos de la B a la F y viendo que la el número de saltos al punto de acceso sería mínimo conectándose a B o C, elige el nodo C como padre porque solo tiene un hijo en ese momento. En segundo caso, el nodo G recibe avisos del nodo raíz A, que es el que menos saltos tiene al *router*, por lo que le elegirá como nodo padre.

A la hora de generar las tablas de enrutamiento para que los nodos puedan enviar y recibir mensajes en la red, cada nodo se guarda las direcciones MAC de los nodos que cuelgan de él y los hijos de él. Sin embargo, nunca tendrán las direcciones de los nodos que tienen por encima. En la Figura 4.9 podemos ver un diagrama de lo que serían las tablas de enrutamiento de los nodos B, C y G. Todos se incluyen a sí mismos y a los nodos que tienen por debajo.

Es muy interesante también la forma que tiene la red de seleccionar los nodos root o

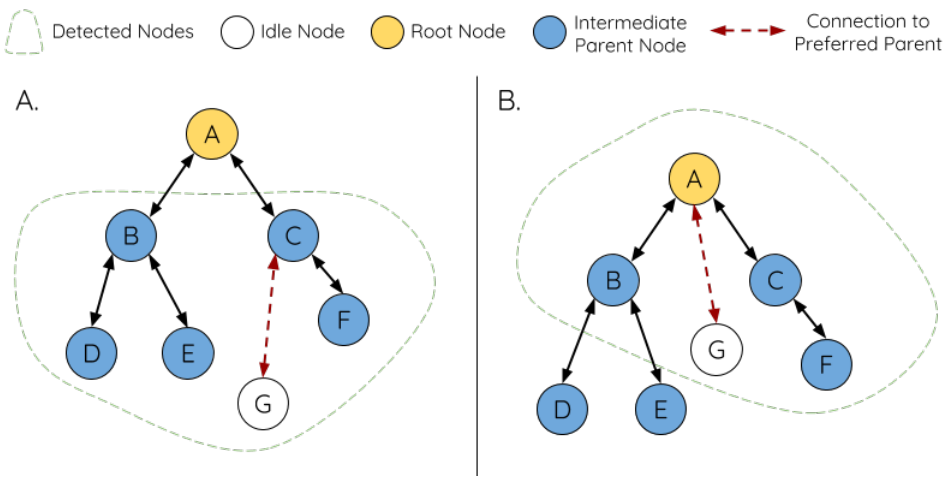

Figura 4.8: *Selección del nodo padre en una red Mesh.*

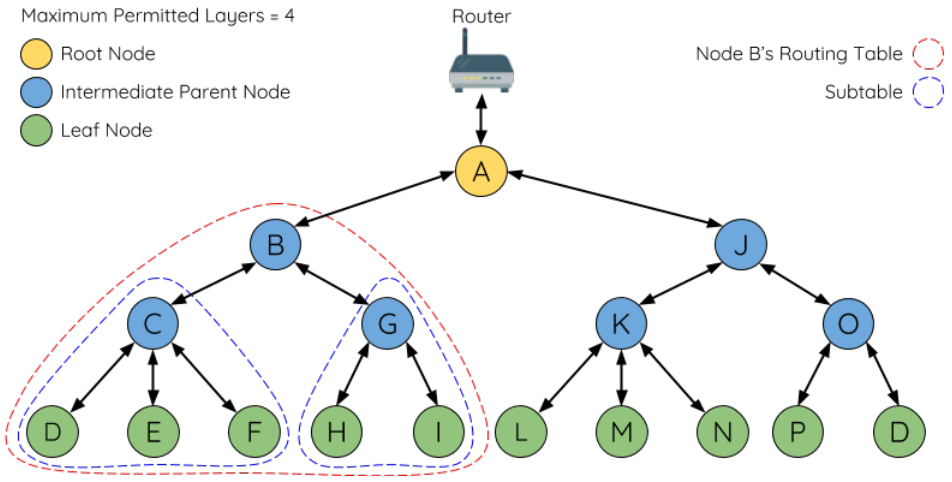

Figura 4.9: *Ejemplo de tabla de enrutamiento en la red Mesh.*

administradores, ya que puede configurarlos el usuario manualmente, o la red puede escogerlos automáticamente evaluando cuál es el mejor candidato. En caso de que lo seleccione el usuario, la red siempre elegirá ese nodo como root y a partir de él se generarán el resto de capas de la red. Sin embargo, cuando es la red la que lo selecciona automáticamente, hace una comparativa entre la potencia de señal del *router* WiFi que le llega a cada uno de los nodos y elige el que tenga la señal más fuerte, que suele coincidir con el que más cerca se encuentre del *router*. Una vez este nodo está configurado y conectado a la red WiFi existente, se generan las demás capas de la malla. De la misma forma que en el caso del

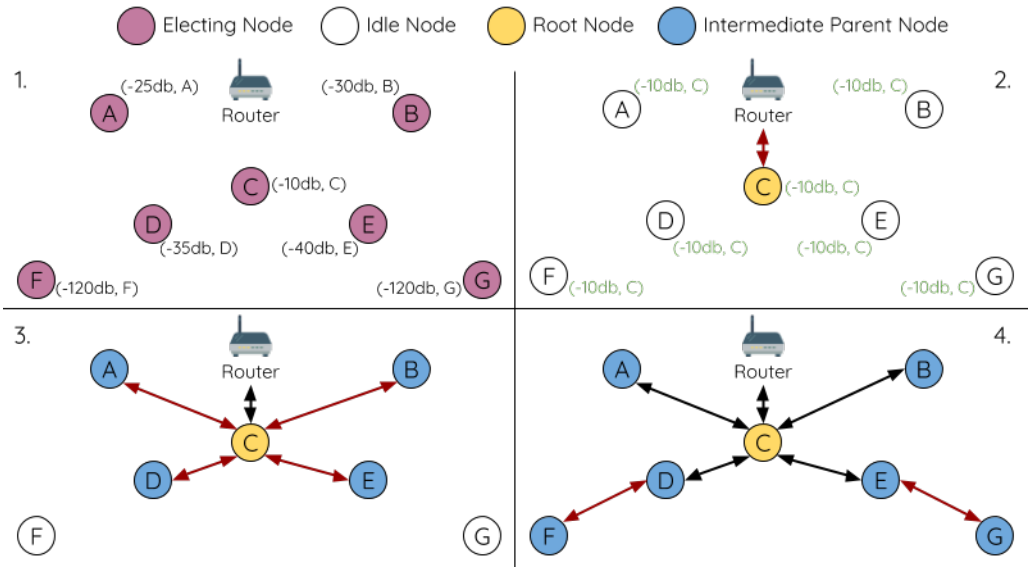

Figura 4.10: *Ejemplo de creación automática de la red Mesh.*

nodo administrador, todos los nodos evalúan cuál es el que mejor calidad de señal tiene y se conectan a ese. La formación de la red aunque sea automática siempre respeta los parámetros fijados por el usuario en cuanto a número máximo de nodos por capa y número de capas. Así, cuando un nodo sabe que se está conectando a la última capa permitida se configura solo como nodo *hoja* para que ningún otro dispositivo se conecte a él.

En la Figura 4.10 se puede ver un ejemplo del proceso de creación de la red Mesh de forma automática. En el paso 1 se ve como los nodos comprueban cuál de ellos es el que tiene mayor potencia de señal del *router*. Con estos datos es el nodo C el que se conecta al *router* y el que actuará de nodo root. A él se conectarán 4 nodos que serán los que más cerca de él se encuentren y finalmente los dos restantes se conectan a los nodos que mejor conexión les ofrezcan, quedando como resultado la red que se ilustra en el paso 4 de la Figura 4.10.

En concreto, en este trabajo se van a probar dos ejemplos de red de malla, uno utilizando protocolo TCP y otro con el protocolo MQTT.

Topología Mesh con TCP

Una red de malla con protocolo de transporte TCP implementa el funcionamiento de la topología que se ha descrito en este apartado, incluyendo la estructura de los mensajes y el procedimiento del intercambio de los mismos descrito en el apartado 4.1.1. Los nodos de la red se conectan entre ellos como cliente/servidor, siendo el cliente el que quiere enviar el mensaje y servidor el que lo recibe. La red que se genere puede estar conectada o no a un *router* que de conexión a la red local con el exterior. Esto puede depender de las necesidades del proyecto en cuestión.

Topología Mesh con MQTT

La aplicación del protocolo MQTT en una red Mesh es muy similar a la de TCP4.2.1, pero incluye una condición adicional que debe cumplirse. Esto es que al menos uno de los nodos de la red debe estar conectado a una red WiFi en la que se encuentre también conectado el broker MQTT que se encargará de la gestión de los mensajes MQTT intercambiados en la red. Una vez que este nodo está conectado con el broker, el resto de los dispositivos de la red funcionan como una red de malla convencional y no es necesario que el resto de los nodos se encuentre dentro del alcance de la red del *router*. El esquema de la estructura de una red Mesh con MQTT como la que se va a implementar en la Sección 5.1.2 puede verse en la Figura 4.11.

4.2.2. Topología estrella

La topología estrella en redes de ordenadores es una de las más utilizadas desde el principio de Internet. Tradicionalmente, se basa en la conexión de distintos ordenadores a través de cables de red a un concentrador, *hub* o *switch*, que es el que realmente da el paso a la red de Internet. Normalmente se utiliza para redes locales que no estén a mucha distancia del *switch*, ya que el despliegue del cable en las redes tradicionales por cable puede suponer un coste alto.

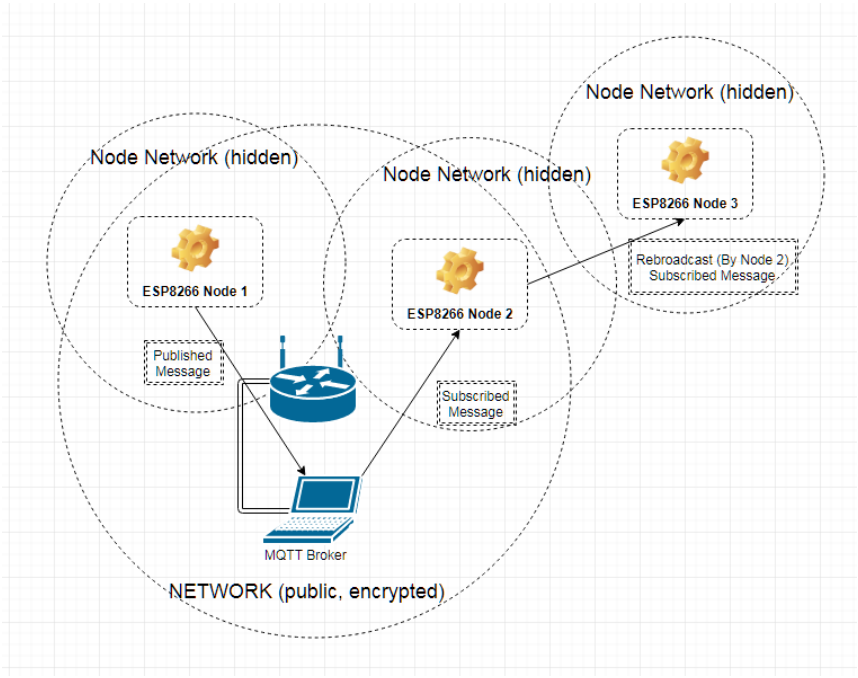

Figura 4.11: *Estructura de una red Mesh con MQTT*[13].

Sin embargo, hoy en día también es muy común que las redes inalámbricas, como puede ser la del WiFi, tengan estructura de estrella. Este tipo de redes se caracterizan también por tener un nodo central que gestiona la conexión de la red con el exterior y al que se conectan todos los dispositivos deseados, como puede verse en la Figura 4.12. Normalmente, el nodo central suele ser un *router* WiFi que conecta la red con Internet.

Las redes que tienen esta topología de estrella no permiten que los dispositivos que la componen sen envíen directamente mensajes entre ellos, todos los paquetes que se envíen deben pasar por el nodo central, y es éste el que se encarga de enviarlos al destinatario real.

El proceso de formación de estas redes es sencillo, ya que los dispositivos que la componen solo necesitan conocer la dirección del *nodo* central y el puerto al que pueden conectarse. Al iniciarse, los nodos buscan si el nodo central está disponible y en caso afirmativo se conectan a él para comenzar el intercambio de mensajes.

Una de las ventajas de este tipo de redes es que en caso de que uno de los nodos de la red falle, el resto de la red no se ve afectada, simplemente se perderán los datos que dependan

Figura 4.12: *Topología estrella*[5].

de ese nodo. Sin embargo, si el que deja de funcionar correctamente es el nodo central la red fallará entera, ya que no se podrá enviar ni recibir ningún paquete.

Igual que para el caso de la topología Mesh y con el objetivo de comparar el funcionamiento de todas las posibilidades, en este trabajo se han implementado dos ejemplo de de topología estrella: uno con protocolo TCP y otro con MQTT.

Topología estrella con TCP

La topología estrella funcionando con protocolo TCP es una de las configuraciones de red más utilizadas. En este caso, uno de los dispositivos de la red debe actuar de servidor que reciba los mensajes y en función de lo que se desee los reenvíe o no. El resto de los nodos de la red son clientes que necesitan conectarse con el servidor para mandar sus mensajes.

Puede ser que el nodo que actúe como servidor sea el mismo dispositivo que actúa de *switch* o *router* en la red, o puede que el nodo servidor se encuentre también conectado a la red a través del *router* y que los paquetes que los clientes quieran enviar tengan que hacer un salto extra antes de llegar al servidor. El servidor cuando reciba el mensaje de los clientes, como se ha visto en la Sección 4.1.1, responderá al cliente con un mensaje de confirmación de recepción.

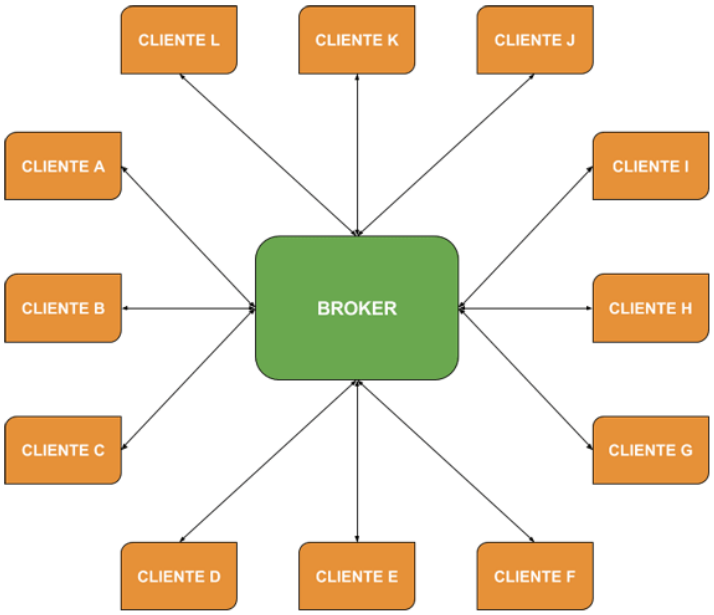

Figura 4.13: *Topología estrella con el broker MQTT en el nodo central*[4].

Topología estrella con MQTT

Para utilizar el protocolo MQTT con la topología estrella es necesario que el broker MQTT que gestioina los mensajes de la red esté conectado, igual que el resto de nodos de la red al mismo *router* y que se encuenrten por tanto, en la misma red local. También puede ser que el propio broker MQTT sea el punto de acceso a la red del resto de nodos, por lo que entonces el broker actuaría como nodo central. El escenario es muy parecido al comentado en el apartado anterior 4.2.2, solo que en este caso el servidor es el broker, ya que el broker es el encargado de reenviar los mensajes y distribuirlos a la red. La estructura de la red en estrella con MQTT puede verse en la Figura 4.13.

Capítulo 5

Análisis y resultados obtenidos

En este apartado se procede al análisis del funcionamiento de distintas redes con los protocolos y topologías descritos en el el Capítulo 4. Se han desplegado cuatro redes y se han recogido mensajes durante el tiempo que han estado funcionando para sacar algunas medidas. Todos los experimentos se han realizado en el mismo escenario para poder comprobar las diferencias entre unos y otros.

En concreto se han realizado cuatro pruebas con dos topologías y dos protocolos combinados: topología *mesh* con protocolos TCP y MQTT y topología estrella con protocolos TCP y MQTT. Para todos los casos se han hecho dos pruebas de funcionamiento: una en el interior de una vivienda para sacar el comportamiento habitual de la red y otra en el exterior, donde se han calculado las distancias máximas entre los dispositivos de la red.

Para el caso del análisis en el interior se han desplegado 8 dispositivos ESP8266 en un domicilio siguiendo el esquema de la Figura 5.1.

En todas las redes los nodos han intercambiado mensajes y se han monitorizado a través del nodo AA que se ha conectado a un ordenador para poder recopilar dichos mensajes. La estructura de los mensajes es muy similar en los cuatro casos, se detallará en los apartados correspondientes a cada uno de ellos. Con estos mensajes se han observado los comportamientos de la red que se describen a continuación.

- Nodos reiniciados: cuántos de los dispositivos que conforman la red se han apagado y vuelto a encender automáticamente durante el tiempo de prueba.

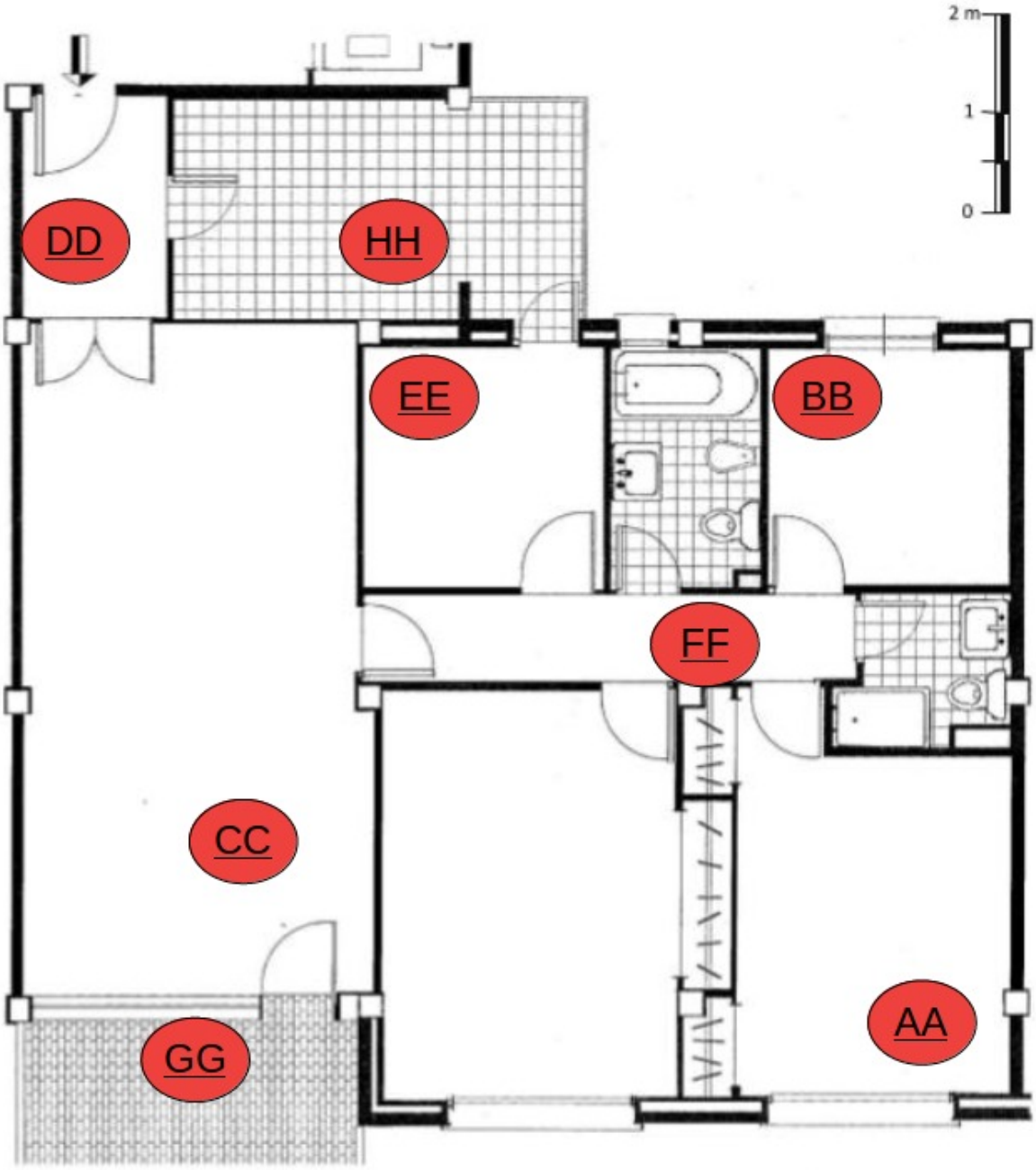

Figura 5.1: *Distribución de los nodos de la red para los experimentos realizados.*

- Nodos que han perdido la conexión: cuántos de los nodos se han desconectado de la red y no han vuelto a sincronizarse.
- Distancia: si la distancia entre los nodos dentro del domicilio ha obstaculizado en algo el intercambio de mensajes.
- Reajuste de la red al conectar y desconectar nodos: cuál ha sido el comportamiento de la red cuando se han conectado y desconectado nodos conscientemente. Esta característica tiene especial interés en el caso de las redes *mesh*, ya que la topología de la red se ajusta automáticamente según los nodos que estén activos y las potencias de señal que haya.
- Potencia de señal: potencia de la red WiFi en cada uno de los nodos que la forman.
- Mensajes perdidos: cuántos mensajes que deberían haber sido enviados por los nodos de la red no han llegado a su destinatario.
- Retardos: el retardo que han sufrido los mensajes desde que se han enviando en uno de los nodos y los ha recibido el receptor.
- Potencia consumida: cuál ha sido la potencia absorbida por los nodos mientras funcionan en cada uno de los casos de prueba. Para este cálculo en concreto ha sido necesario medir el consumo real de los dispositivos ESP8266 durante su funcionamiento y para ello se han utilizado dos dispositivos de medida: un amperímetro y un voltímetro. En todos los casos se han conectado estas herramientas de la misma forma y se han realizado las mismas dos pruebas. Primero se ha hecho la prueba con dos dispositivos conectados en paralelo y después con uno solo, ya que como los números obtenidos dan un valor muy bajo, se dispone de varias medidas de referencia. De esta forma, se han realizado cuatro tipos de circuitos físicos para uno y dos dispositivos con el amperímetro y el voltímetro conectados donde ha sido necesario. Solo se disponía de un multímetro que ha sido el que ha funcionado de voltímetro y de amperímetro, por

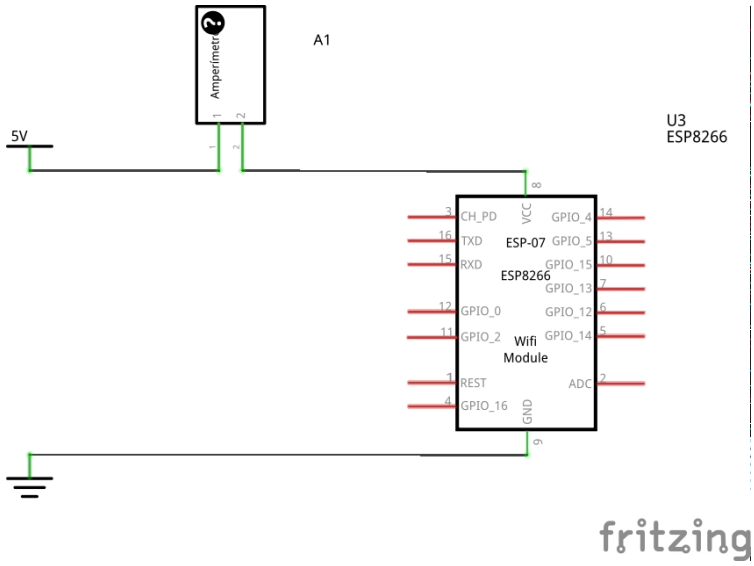

Figura 5.2: *Esquema de la medición del consumo de corriente de un dispositivo ESP8266.*

lo que se han tenido que tomar cada una de las medidas por separado. El voltímetro mide la tensión entre las bornas de alimentación de los chips, por lo que se conecta en paralelo con las mismas, como puede verse en la Figura 5.4 para un solo chip y en la Figura 5.5 para dos. El amperímetro sin embargo, se conecta en serie con las bornas de alimentación de los ESP, ya que la corriente se mantiene dentro de la malla del circuito. El esquema de esta medición puede verse en la Figura 5.2 para un solo chip y en la Figura 5.3 para dos.

Para calcular la distancia entre los dispositivos de la red se han hecho pruebas con los cuatro tipos de redes en un espacio abierto y en línea recta. El lugar donde se han tomado las medidas es un pueblo de Soria en el que no hay interferencia de redes inalámbricas, por lo que los datos obtenidos puede que sean más altos de lo que se obtendría en una gran ciudad.

Una vez se recopilaron los datos se realizaron algunos cálculos con ellos en hojas de Cáculo de Google cuyos enlaces pueden encontrarse en el Apéndice A de este mismo documento. Todo el código de los programas utilizados para el análisis de las redes se encuentra en el repositorio de Github que se indica en el Apéndice B, dentro de las carpetas *estrella* y *mesh*.

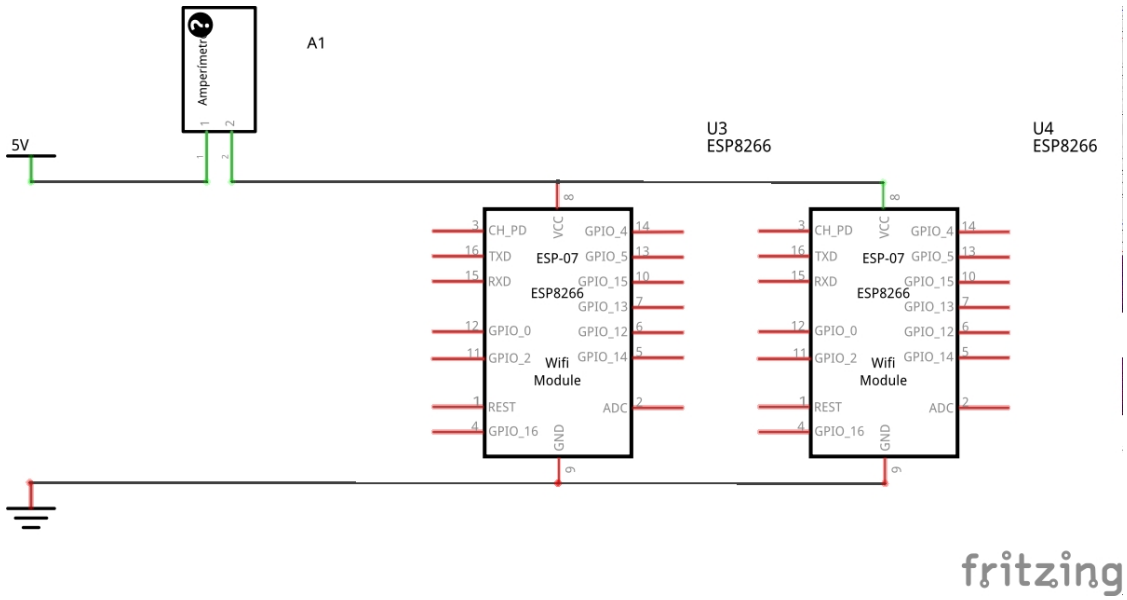

Figura 5.3: *Esquema de la medición del consumo de corriente de dos dispositivo ESP8266.*

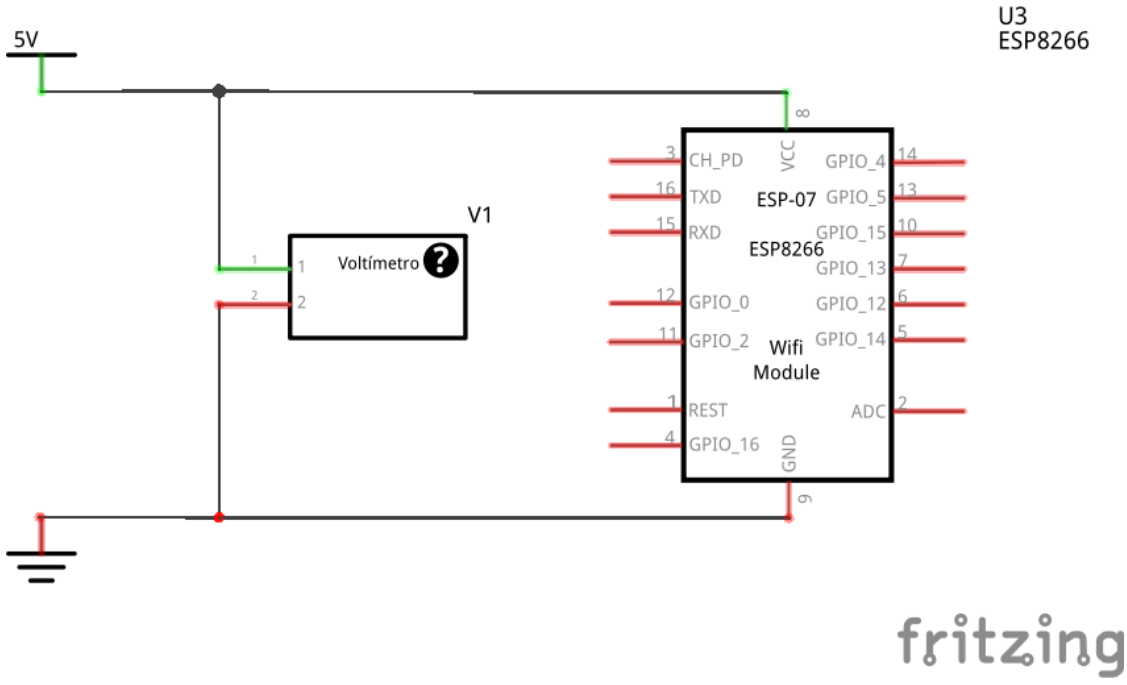

Figura 5.4: *Esquema de la medición del voltaje suministrado al circuito para un dispositivos.*

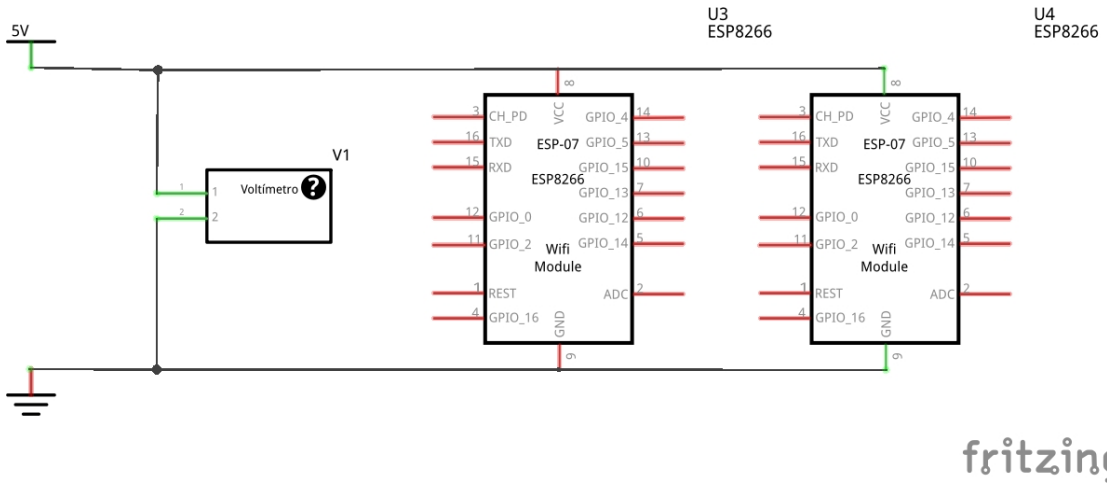

Figura 5.5: *Esquema de la medición del voltaje suministrado al circuito para dos dispositivos.*

5.1. Topología *mesh*

5.1.1. Protocolo TCP

El protocolo de control de transmisión, más conocido como TCP, se basa en la transmisión de paquetes de forma segura y comprobando que el destinatario lo ha recibido.

Para probar una red *mesh* que lo utilice, se ha utilizado el Framework Arduino, con la librería *painlessMesh*[11], adaptando uno de los ejemplos que proporciona el autor: *basic.ino*. La librería permite la creación automática de una red de malla en la que los nodos detectan los otros nodos que tienen a su alcance y establece conexiones según las condiciones explicadas en el apartado 4.2.1. Con el objetivo de proteger la red implementada, se establecen unas credenciales de acceso que se han definido en el código y que son un prefijo de red, una contraseña y el puerto.

Análisis en interior

Con el objetivo de analizar el funcionamiento de la red mesh dentro de un domicilio se ha realizado una prueba con 8 dispositivos ESP8266 conectados e intercambiando mensajes.

Para ello, se ha utilizado uno de los ejemplos incluidos en la librería mencionada ante-

riormente, PainlessMesh. Este primer ejemplo es muy sencillo, pero se han modificado los mensajes enviados y recibidos por los nodos para sacar datos sobre la red desplegada.

Cada nodo envía un mensaje con su nombre, un número que cuenta la cantidad de mensajes que se han mandando desde que el nodo se conectó a la red, la potencia de la señal WiFi en ese momento y el tiempo del en el que se envía. En la segunda fila de la Tabla 5.1 puede verse la estructura final del mensaje enviado. Al recibir un mensaje, cada nodo le añade su nombre, la hora a la que se recibe y la potencia de señal en ese instante. Uno de los nodos se ha conectado al monitor serie del ordenador para poder acceder a estos mensajes y poder analizarlos, por lo que la estructura final que se ha analizado es la que incluye los datos de los nodos emisor y receptor, que se muestra en la Tabla 5.1. Además, la topología mesh, cada vez que hay un nodo nuevo que se conecta o desconecta a la red, se detecta automáticamente y también lo muestra por pantalla.

ID del nodo receptor	potencia señal receptor	hora de recepción	
ID del nodo emisor	número de mensaje	potencia señal emisor	hora de envío

Tabla 5.1: *Formato del mensaje final que muestran los nodos de la red.*

Para sacar todos estos datos, se han utilizado funciones proporcionadas por la librería painlessMesh. La potencia de la señal en cada momento se calcula con la función *RSSI* y la hora del momento en que se envíe con *getNodeTime*, que proporciona el tiempo de cada nodo dentro de la red. Lo interesante de esta última función es que la propia red sincroniza los relojes de los dispositivos, por lo que ha sido muy útil para el cálculo de los retardos entre envíos y recepciones de mensajes. El número de mensaje simplemente se hace con un contador que aumenta en uno cada vez que se envía un mensaje. Cada uno de los nodos envía un mensaje cada 8 segundos.

Con este programa implementado, y una vez se ha cargado el programa en todos los dispositivos, se procede al análisis. En esta primera prueba se han mantenido ocho nodos conectados simultáneamente durante una hora. Los dispositivos se han esparcido por un domicilio, de tal forma que tuvieran paredes y otros objetos que pudieran interferir en la

RX node	RX rssi	RX time	TX node	TX n of message	TX rssi	TX time
AA	-69	390122819	HH	37	-61	390094721
AA	-68	392502232	GG	2	-47	392490432
AA	-69	393900657	BB	49	31	393793389
AA	-68	394305126	CC	3	-68	394309491
AA	-69	395009978	EE	41	-75	394990316
AA	-69	395596600	DD	36	-70	395572081
AA	-69	395697663	FF	47	-68	395689921

Tabla 5.2: *Ejemplo de secuencia datos recogidos durante el experimento en la topología Mesh con TCP.*

transmisión de paquetes. La distribución es la que puede verse en la Figura 5.1.

Con esta colocación de los nodos, y una vez recogidos los datos, se han analizado con tablas de Hojas de Cálculo de Google (Anexo A). Un ejemplo de los datos con los que se ha trabajado puede verse en la Tabla 5.2.

Siguiendo los resultados de estos cálculos, se han observado los comportamientos que se muestran a continuación.

- **Nodos reiniciados:** Ninguno de los nodos se ha reiniciado, todos han funcionado correctamente. Cada uno de ellos ha llegado a mandar casi 3000 mensajes.
- **Nodos que han perdido la conexión:** Ninguno de los nodos ha perdido la conexión con su nodo padre.
- **Distancia:** Durante el tiempo que ha estado funcionando la red, ninguno de los nodos se ha visto afectado por la distancia.
- **Reajuste de la red al conectar y desconectar nodos:**
  - Al desconectar el nodo FF que se encontraba en el pasillo (Figura 5.1), la red ha detectado un cambio. Durante un par de minutos se han recibido mensajes únicamente de cinco nodos, por lo que al desconectarlo, ha habido un nodo que ha perdido la conexión con la BB y por tanto con la red. A continuación, la red

ha vuelto a detectar cambios y el nodo del que no se recibían mensajes por la desconexión del nodo FF, la red se ha recuperado. Se reciben mensajes de seis de los 8 nodos, ya que el FF sigue desconectado.

- Se ha vuelto a encender el nodo FF y la red ha detectado cambios de nuevo. El nodo FF se ha reiniciado y conectado con su nodo padre, por lo que ya se reciben mensajes de los siete nodos.

Es interesante comentar, que a pesar de los cambios en la red debido a uno de los nodos, el resto no ha experimentando ninguna anomalía. Todos siguen enviando los números de mensajes sin interrupción, incluso el que perdió la conexión y tuvo que conectarse a otro nodo, a excepción de FF que al haberse reiniciado ha empezado de cero.

- **Potencia de la señal:** La potencia de señal la miden cada uno de los nodos cada vez que envíen o reciben un mensaje. En esta topología cada uno de los nodos genera una pequeña red WiFi a la que se conectan el resto de los nodos, por lo que los datos obtenidos se corresponden con la topología de los nodos en ese instante, ya que son las intensidades de la conexión de los nodos con su nodo padre. Según las especificaciones de la librería cada nodo padre puede tener como máximo 4 hijos y el esquema de la topología de la red durante la prueba puede verse en la Figura 5.6. En este caso se está monitorizando la red desde el nodo AA, pero esto no quiere decir que este sea el nodo *root* de la red, por lo que el esquema no es fiel a la realidad. Los datos finales de la potencia media de señal recibida por cada nodo en dBm son los que pueden verse en la Tabla 5.3. Según estos datos, podría deducirse que el nodo padre es el nodo BB, que se ve que detecta a su nodo padre una potencia de 31 dBm y esta potencia es imposible de alcanzar con estos dispositivos, por lo que se asume que es el valor de 0 dBm para el sistema. Viendo los datos de la tabla, puede deducirse que el resto de la red se genera a partir de este nodo BB. Por tanto, al nodo BB estarían conectados

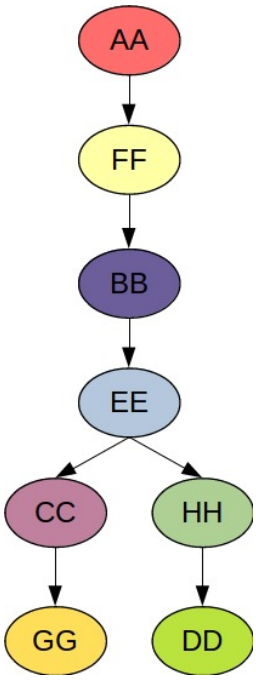

Figura 5.6: *Esquema de la topología de la red durante la prueba.*

los nodos EE y FF, y según la colocación del plano de la Figura 5.1 el nodo FF está más cerca de BB que EE, por lo que la potencia de señal en el nodo FF debería ser mayor. En este caso se cumple. Al nodo FF se conecta el AA y según los datos, hay una potencia de señal coherente de -77,45 dBm. Al nodo EE se conectan los nodos CC y HH, estando CC mucho más lejos de su nodo padre que HH. Por tanto la potencia media de señal es mucho más fuerte en el nodo HH. A su vez, los nodos HH y CC tienen un hijo cada uno: DD y GG respectivamente. GG está mucho más cerca de su padre que DD, por lo que debería tener una potencia de señal mayor y según los datos de la tabla, se cumple.

- **Mensajes perdidos:** Para sacar los mensajes perdidos en la red durante el experimento se han ordenado todos los mensajes por nodo y número de mensaje enviado. Con esta colocación se han buscado los mensajes que no hayan sido números consecutivos y se ha comprobado que no ha habido ningún salto. Todos los mensajes han sido enviados y recibidos por los nodos de la red, no ha habido ninguna pérdida.

Nodo	Potencia (dBm)
AA	-77,45272119
BB	31
CC	-76,48883929
DD	-69,60215054
EE	-79,50537634
FF	-72,08172043
GG	-33,19866071
HH	-46,06717003

Tabla 5.3: *Potencia media de señal de cada nodo de la red.*

- **Retardos:** Gracias a la función *getNodeTime* facilitada por la librería PainlessMesh, se ha podido calcular el retardo de los mensajes desde que se envían hasta que se reciben con una precisión de milisegundos. Como la red mantiene sincronizados todos los nodos, basta con restar el tiempo de recepción y el de emisión para sacar la latencia en la recepción del mensaje. A simple vista observando los datos obtenidos parece que el retardo de mensajes está en torno a unas decenas de milisegundos, pero la media final calculada a partir de la diferencia entre tiempos de recepción y transmisión ha sido de 247,39 milisegundos, inferior a un segundo.

- **Potencia consumida:** La potencia consumida por los nodos de la red durante el funcionamiento de la misma es un dato interesante de conocer, de cara a la alimentación de las placas en caso de un despliegue real. Para calcularla se ha hecho un circuito físico con un multímetro actuando primero como amperímetro y después como voltímetro para poder calcular manualmente con la fórmula $P=V*I$ la potencia consumida por los dispositivos. Al tratarse de medidas muy pequeñas se han hecho pruebas con uno y dos ESP8266 colocados en paralelo, como se ha explicado en el Capítulo 5. Se han hecho las dos medidas para los dos circuitos y se ha obtenido que el voltaje es constante con un valor de *4,9 V* y la corriente ha tomado los valores que pueden verse en la Tabla 5.4. En esta tabla también se encuentra el valor final de la potencia consumida por las placas, siendo 62,72 μW para una placa y de 134,75 μW para dos.

	Corriente (uA)	Potencia (μW)
Una placa	12,8	62,72
Dos placas	27,5	134,75

Tabla 5.4: *Potencia consumida por uno y dos nodos de la red en topología Mesh con TCP.*

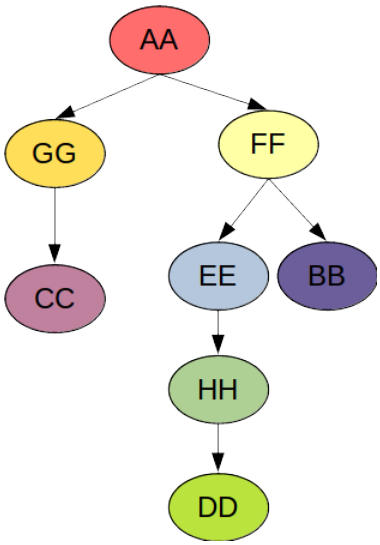

Figura 5.7: *Esquema de la topología de la red con todos los nodos conectados.*

Con el objetivo de comprobar si la red Mesh ha funcionado como tal, se ha utilizado la función *subConnectionJson()* de la librería *PainlessMesh*. Con esta función se puede acceder al esquema de la conexión de los nodos y para ello ha añadido que el programa de prueba muestre por la consola el resultado de llamarla cada vez que cambian las conexiones de la red. La librería utilizada permite únicamente 4 ESP8266 conectados a cada nodo, por lo que la topología de la red se creará en base a esta condición. Para esta prueba, se han conectado y desconectado algunos de los nodos para ver la respuesta de la red en cada momento.

La nueva distribución en el espacio de los nodos es la que puede verse en la Figura 5.1. Y la topología obtenida con dicha distribución es la que se muestra en el esquema de la Figura 5.1. El nodo AA es el que se encuentra conectado al ordenador y desde el que vemos las trazas en el monitor serie, por lo que en todos los casos será el que esté en la posición más alta del árbol construido por la red.

El primer nodo que se desconecta es el nodo EE, e inmediatamente la red lo detecta y por

aparece una nueva topología idéntica a la que estaba anteriormente, pero sin éste nodo. Sin embargo, pasados unos segundos la red se reajusta y es obtiene la topología que se observa en la Figura 5.8.

A continuación, se desconecta el nodo FF, que es el que más cerca está del nodo padre AA. Igual que en el caso anterior, la red lo detecta inmediatamente pero esta vez tarda unos segundos más que antes en reajustarse. La primera topología que muestra la red es la que puede verse en la Figura 5.9, pero sin el nodo BB, por lo que solo se veían 5 nodos conectados a la red. Pasados varios segundos el nodo BB no aparecía, por lo que ha tenido que reiniciarse y finalmente se ha conectado al nodo AA tal como se muestra en la Figura 5.9.

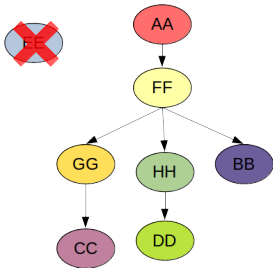

Figura 5.8: *Esquema de la topología de la red sin el nodo EE.*

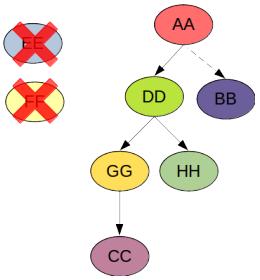

Figura 5.9: *Esquema de la topología de la red sin los nodos EE ni FF.*

Al volver a conectar los nodos EE y FF a la red, la topología resultante vuelve a ser la que se muestra en la Figura 5.7.

Análisis en exterior

Para esta prueba se han realizado distintas pruebas mezclando un entorno cerrado, una casa, y un espacio abierto, la calle. Los nodos se han ido cambiando de sitio y conectando y desconectando en ciertos momentos para ver el comportamiento de la red en cada caso. En los mapas, puede verse en color naranja o rojizo los espacios que se corresponden con edificios y en color verde o blanco los que son espacios abiertos.

En el primer caso se ha monitorizado la red desde uno de los nodos que se encontraba

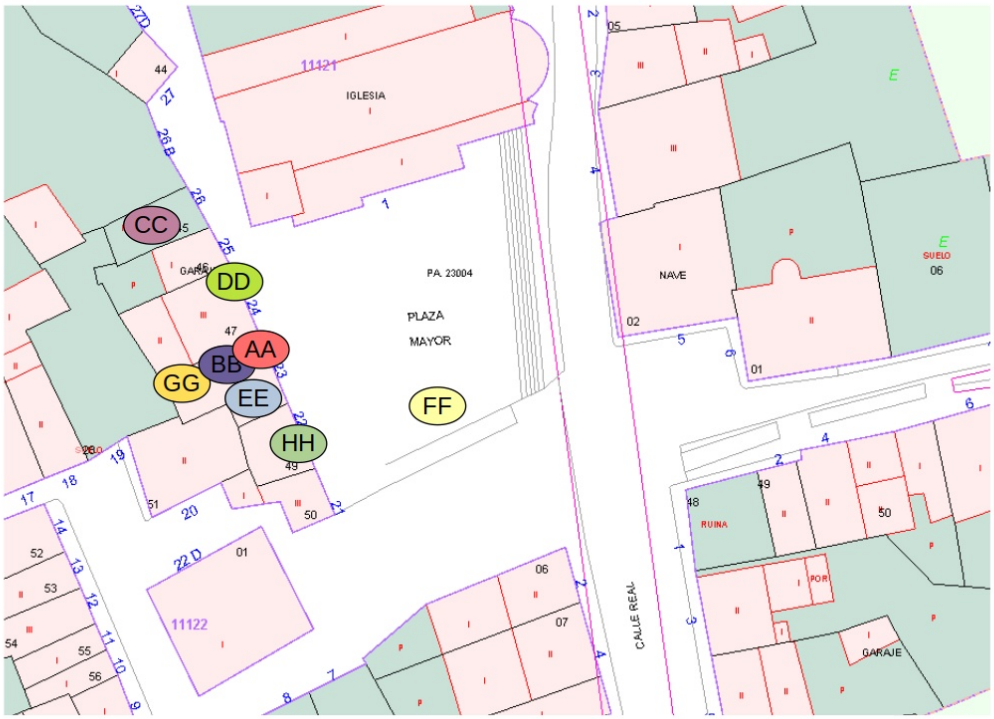

Figura 5.10: *Distribución de los nodos en el espacio.*

dentro del domicilio y se ha obtenido que la topología de la red con la distribución de la Figura 5.10 es la que se ve en la Figura 5.11. La función *subConnectionJson()* muestra la red tal y como la recibe el nodo desde la que la que se está viendo. Dado que el ordenador está conectado al nodo EE, podemos ver que actúa de nodo raíz. Los nodos que se encuentran en el interior del domicilio son los nodos BB, GG y EE, el resto se encuentran en el exterior. Es importante destacar cómo el nodo EE se conecta únicamente con el nodo GG y es éste el que se conecta con el CC y el HH para poder mandar los paquetes a través de toda la red.

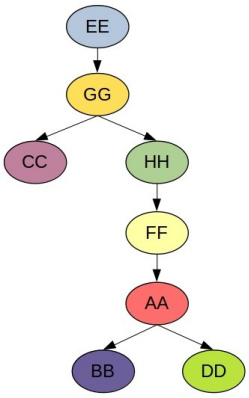

Figura 5.11: *Esquema de la topología de la red mesh de la Figura 5.10*

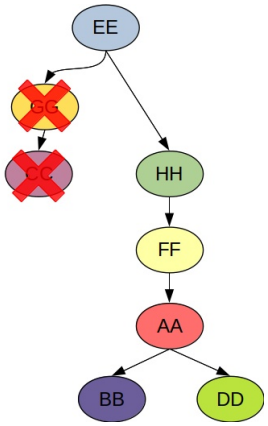

Figura 5.12: *Esquema de la topología de la red mesh con GG desconectado correspondiente al plano de la Figura 5.13*

Para comprobar el correcto funcionamiento de la topología *mesh* se ha procedido a desconectar el nodo GG que actúa de nexo entre el nodo que monitoriza la red y el resto. En esta ocasión vemos que el mapa queda como el de la Figura 5.13 y la topología como la de la Figura 5.12. Puede verse que se ha perdido la conexión tanto con el nodo GG como con el CC, que accedía a la red a través del nodo desconectado y por tanto ahora sólo hay una rama de nodos conectados, que es la misma que en el caso anterior.

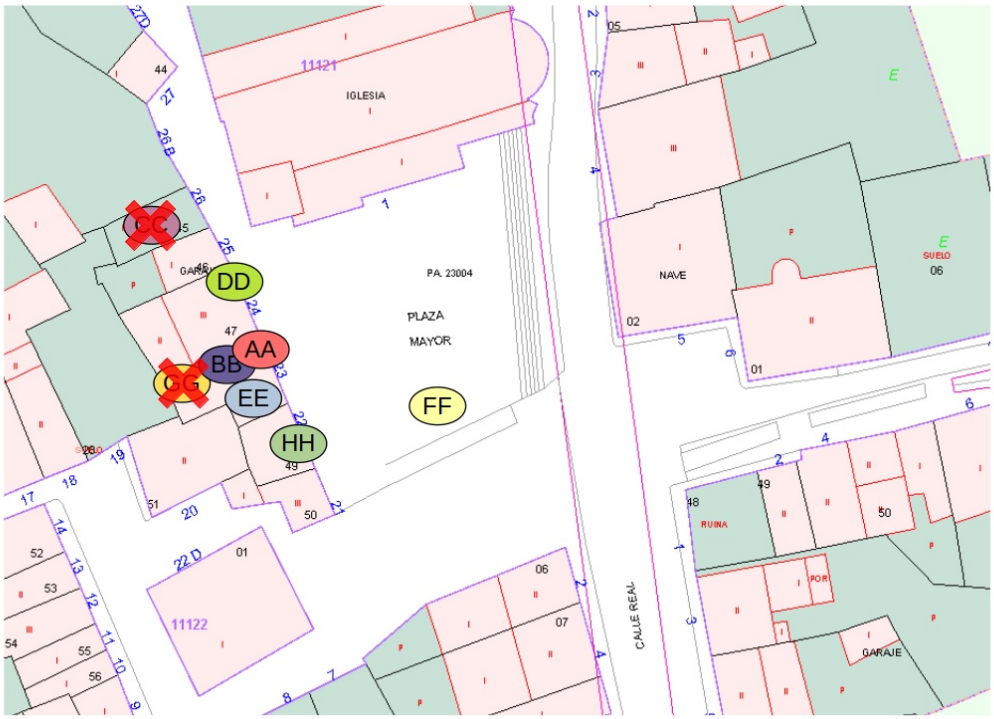

Figura 5.13: *Distribución de los nodos en el espacio con GG desconectado.*

A continuación, se desplazan los nodos FF y DD para ver si es posible conseguir reconectar el nodo CC. La nueva distribución es la de la Figura 5.14 y la topología resultante que han establecido los nodos se ve en la Figura 5.15. Se comprueba que al mover los dos nodos se ha perdido además la conexión con el nodo HH y que la de CC no se recupera.

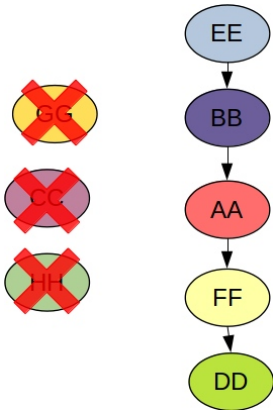

Figura 5.15: *Esquema de la topología correspondiente a la distribución de los nodos de la Figura 5.14.*

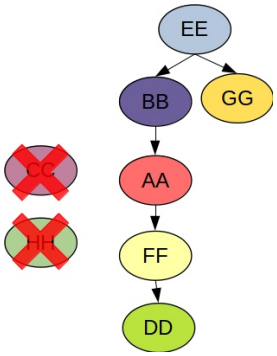

Figura 5.16: *Esquema de la topología correspondiente a la distribución de los nodos de la Figura 5.14, una vez se ha conectado GG.*

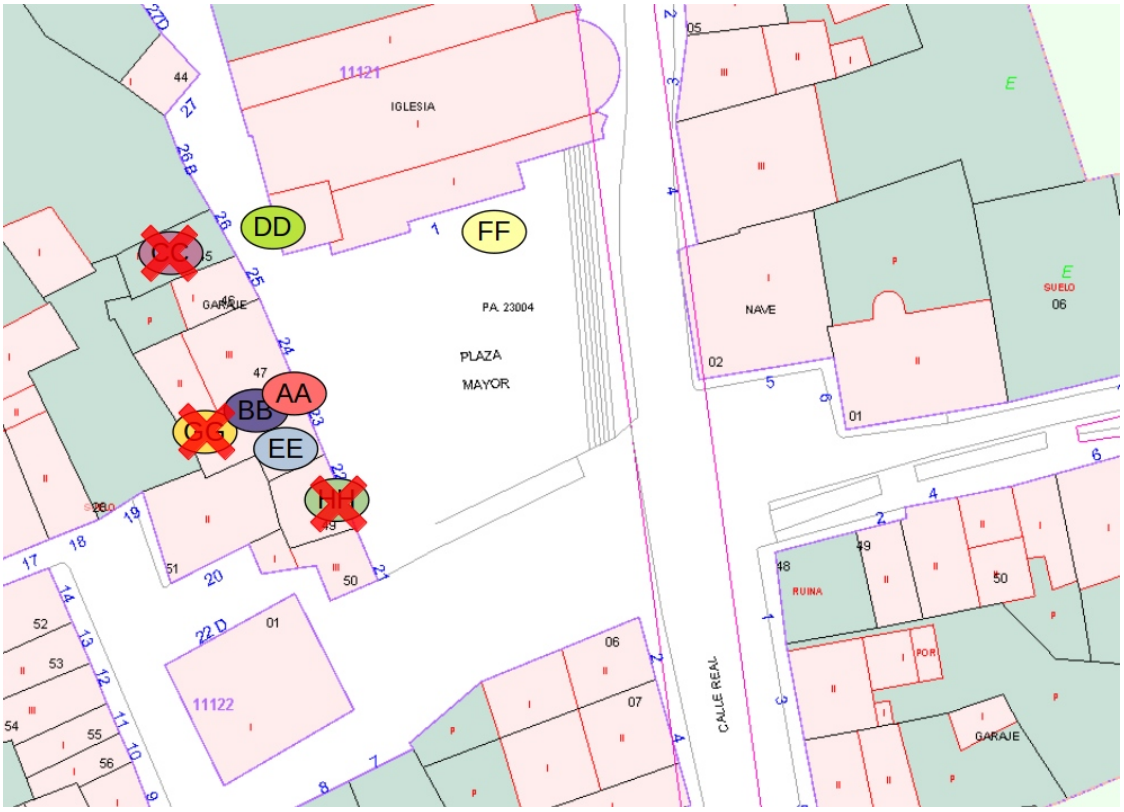

Figura 5.14: *Nueva distribución de los nodos en el espacio.*

Al volver a encender el nodo GG se conecta como una nueva rama que cuelga del EE (Figura 5.16), pero se sigue sin recuperar la conexión de los nodos CC y HH. Al resetear el

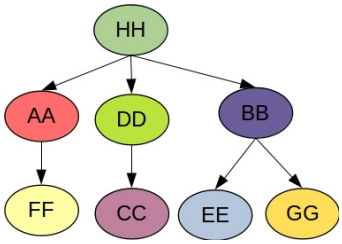

Figura 5.17: *Esquema de la topología inicial de la distribución de los nodos de la Figura 5.19 vista desde HH.*

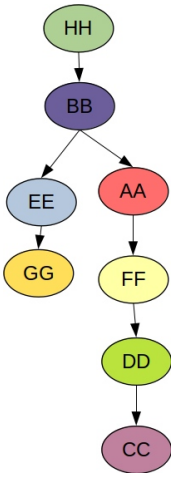

Figura 5.18: *Esquema de la topología final de los nodos vista desde HH, correspondiente al plano de la Figura 5.19*

nodo CC, se vuelve a conectar a la red.

Para comprobar el funcionamiento de la red cuando se resetea el nodo padre se ha probado a desenchufar el nodo EE y a conectar el ordenador para ver los *logs* de las conexiones en el nodo HH. Ahora vemos que están todos los nodos conectados y que la forma de la red es totalmente distinta a la anterior, teniendo tres ramas que salen de HH, como puede verse en la Figura 5.17.

Finalmente, se han alejado dos de los nodos para comprobar el alcance de los mismos y la distribución del mapa es como la que se ve en la Figura 5.19. En este plano puede verse que del nodo DD sale una flecha. Esta indica que durante este análisis el nodo DD se desplazó para comprobar los efectos que tenía en la topología. La forma de la red vista desde el nodo HH es la que se representa en la Figura 5.18 y se comprueba que al mover el nodo DD la topología de la red no cambia en absoluto. Todos los nodos siguen con las mismas conexiones.

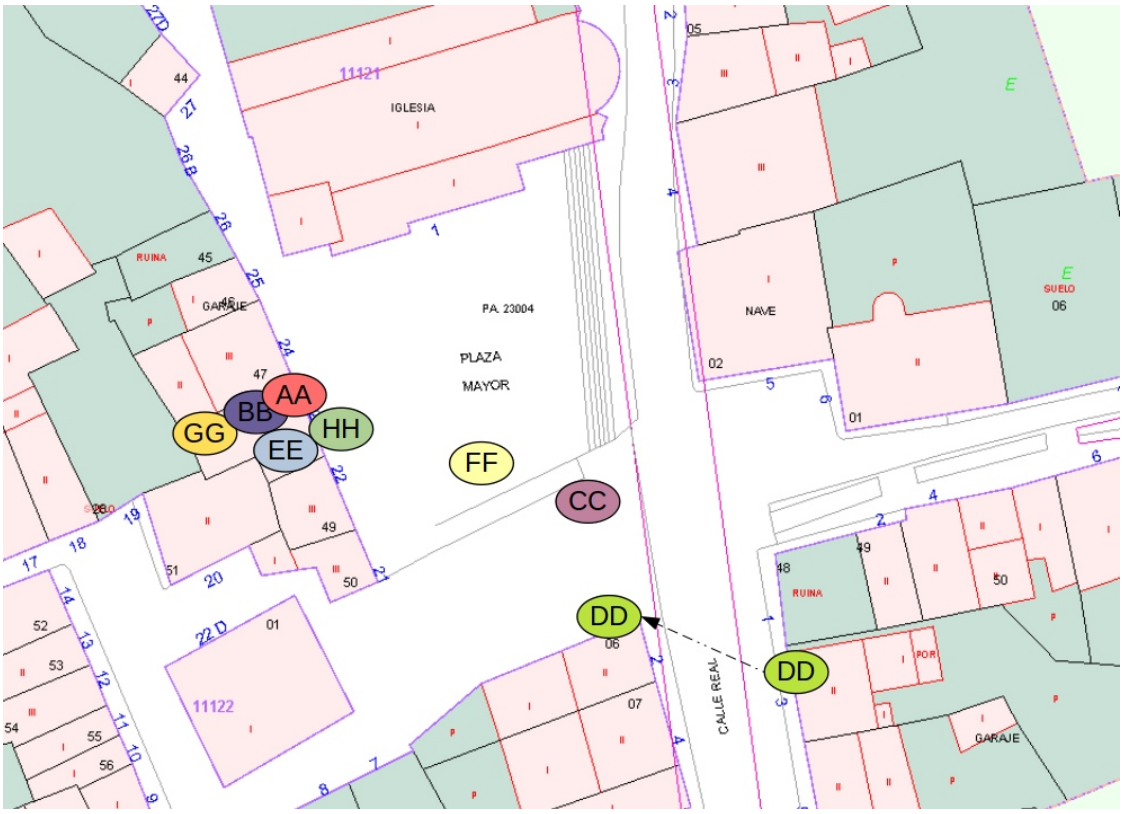

Figura 5.19: *Nueva distribución de los nodos en el espacio.*

Análisis de distancia

Por otro lado, para comprobar la distancia máxima entre dos dispositivos se han desconectado todos menos los nodos AA y BB y se ha obtenido que la conexión entre ambos se pierde a unos 200 metros en línea recta y en exterior, según puede verse en el plano de la Figura 5.20. Dejando el nodo AA conectado al ordenador y desplazando el BB puede observarse en la terminal serie del nodo AA de la Figura 5.21, que la red detecta que hay cambios en la conexión. Puede verse también que desde que el nodo BB pierde la conexión hasta que la recupera. Hay 7 mensajes que se han perdido, ya que el chip ha continuando enviándolos aunque no estuviera conectado a la red.

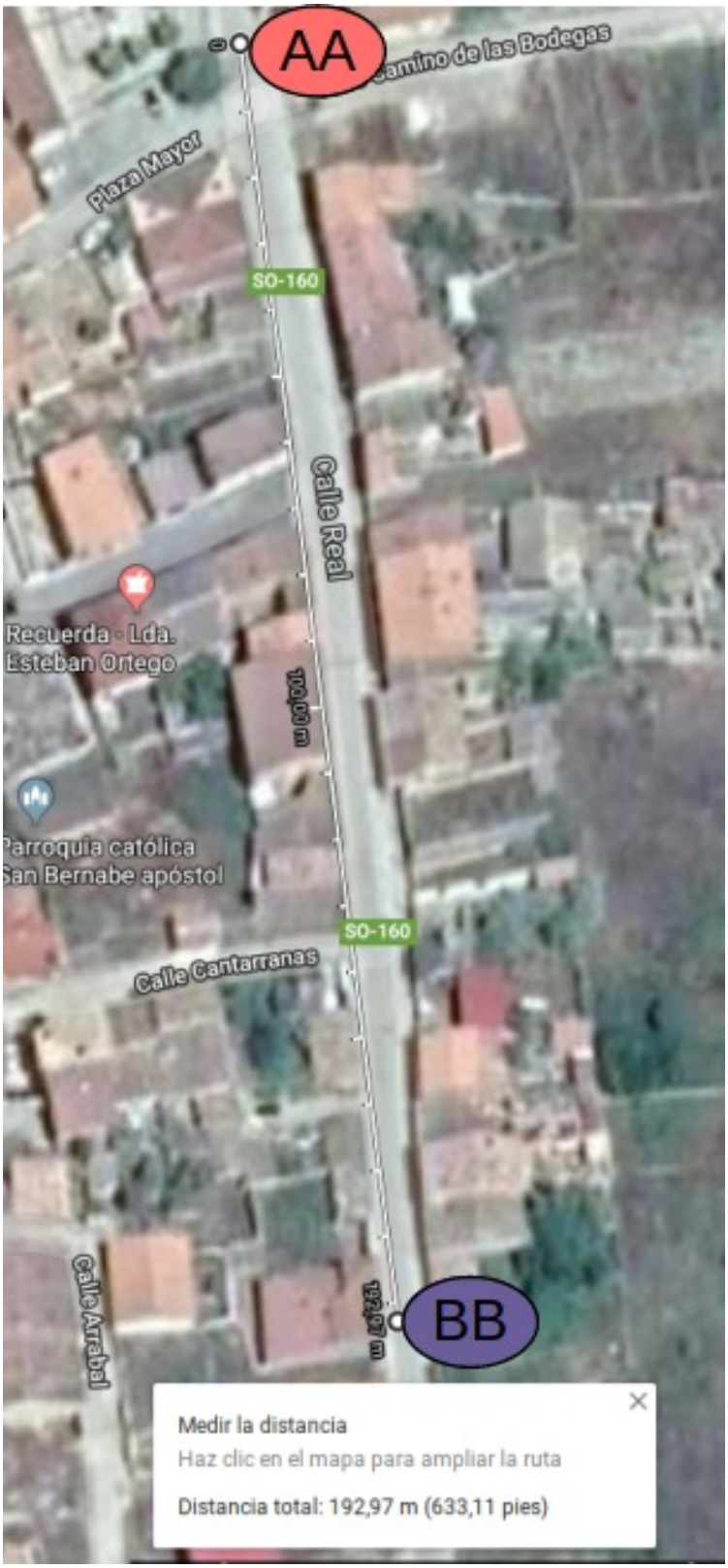

Figura 5.20: *Distancia máxima entre dos nodos de la red Mesh con TCP.*

```
startHere: Received from 977122390 msg=Hello -> 110 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 111 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 112 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 113 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 114 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 115 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 116 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 117 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 118 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 119 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 120 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 121 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 122 <- from node 977122390
Changed connections
Connections in NEtwork = {"nodeId":3809054052
Changed connections
Connections in NEtwork = {"nodeId":3809054052,"subs":[{"nodeId":977122390}]
--> startHere: New Connection, nodeId = 977122390
startHere: Received from 977122390 msg=Hello -> 129 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 130 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 131 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 132 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 133 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 134 <- from node 977122390
startHere: Received from 977122390 msg=Hello -> 135 <- from node 977122390
```

Figura 5.21: *Monitor serie del nodo AA cuando pierde la conexión con BB.*

5.1.2. Protocolo MQTT

El protocolo MQTT es uno de los más utilizados en IoT, por lo que es interesante analizar el comportamiento del mismo en una red de topología Mesh.

Igual que en el caso anterior, se ha realizado una prueba de funcionamiento del mismo utiliznado el Framework de Arduino para los dispositivos ESP8266 que han funcionado como nodos de la red. En concreto, se ha implementado la red con ayuda de la librería ESP8266MQTTMesh[12], que proporciona las funciones necesarias. Como se explica en las especificaciones de la misma, es necesario que uno de los nodos esté conectado a la red de un *router* WiFi y que en ésta se encuentre funcionando el broker MQTT necesario para el funcionamiento de este protocolo, como se ha explicado en la Sección 4.1.2. El resto de los nodos de la red puede conectarse entre ellos, teniendo como máximo 4 nodos hijos

conectados. Los mensajes intercambiados en la red viajan de tal forma que lleguen al broker MQTT.

Es necesario proporcionarle a todos los nodos las credenciales de la red WiFi en la que está conectado el broker y además se ha incluido una contraseña para todos los nodos que proteja la red de malla implementada.

Análisis en interior

Para realizar el experimento con la topología Mesh y el protocolo MQTT se ha utilizado como base uno de los programas de ejemplo que viene con la librería ESP8266MQTTMesh. El programa consiste en mandar mensajes sencillos cada 8 segundos a la red con datos sobre el estado de las conexiones en ese momento.

Cada nodo envía su nombre, un ID de mensaje que identifique el número de mensajes mandados por cada dispositivo, la potencia de señal WiFi en la transmisión y la hora de envío del mensaje. Al recibirlo, los nodos muestran por el monitor serie el mensaje recibido, y además los datos del receptor: el nombre, la potencia de señal del receptor, la hora de recepción y el topic MQTT al que está suscrito. La estructura de los mensajes enviados puede verse en la segunda fila de la Tabla 5.5 y la tabla completa se corresponde con los mensajes recopilados para el análisis de la red.

Topic MQTT	ID del nodo receptor	potencia señal receptor	hora de recepción
ID del nodo emisor	número de mensaje	potencia señal emisor	hora de envío

Tabla 5.5: *Formato del mensaje en los nodos receptores de la red Mesh MQTT.*

Para acceder a estos datos se han utilizado algunas funciones proporcionadas por la librería mencionada anteriormente, ESP8266MQTTMesh, pero para otros ha sido necesario añadir otras librerías. Este ha sido el caso de las horas de envío y recepción del mensaje, ya que los nodos llevan un contador de tiempo desde que se encienden, pero no están sincronizados unos con otros. Por ello, se ha añadido un cliente NTP, con las librerías WiFiUdp que forma parte de ESP8266WiFi[6] y NTPClient[2], para poder acceder a la hora

topic	RX node	RX rssi	RX time	TX node	TX n of msg	TX rssi	TX time
[TFM/test/2/]	AA	-68	0:01:17	DD	28	-64	0:03:46
[TFM/test/2/]	AA	-68	0:01:20	CC	32	-50	0:04:20
[TFM/test/2/]	AA	-68	0:01:21	GG	33	-52	0:04:26
[TFM/test/2/]	AA	-68	0:01:21	EE	24	-60	0:03:14
[TFM/test/2/]	AA	-69	0:01:22	FF	18	-76	0:02:28
[TFM/test/2/]	AA	-68	0:01:23	BB	17	-78	0:02:20
[TFM/test/2/]	AA	-69	0:01:24	AA	10	-68	0:01:24
[TFM/test/2/]	AA	-68	0:01:24	HH	25	-77	0:03:29

Tabla 5.6: *Ejemplo de secuencia completa de mensajes recibidos.*

proporcionada por la red y así comparar los tiempos de recepción y emisión. El servidor NTP proporciona precisión de segundos, por lo que el cálculo de los retardos no será demasiado preciso.

Para esta prueba, se han distribuido 8 nodos en un domicilio, como se vio en la Figura 5.1 y se ha dejado que la red intercambiara datos durante una hora.

En principio el nodo AA es el encargado de monitorizar la red, por lo que se han recolectado los mensajes que muestra por el monitor serie y a continuación se han analizado con tablas de Hojas de Cálculo de Google. Un ejemplo de una secuencia completa de los datos trabajados puede verse en la Tabla 5.6.

Una vez realizado el estudio, se han observado los comportamientos que se explican a continuación.

- **Nodos reiniciados:** No se ha reiniciado ningún nodo durante todo el experimento. Todos han funcionado correctamente.

- **Nodos que han perdido la conexión:** Ningún nodo ha perdido la conexión con la red.

- **Distancia:** En este ejemplo realizado en interior ningún nodo se ha visto afectado por la distancia. Más adelante se hará una prueba específica de distancia máxima entre nodos, en la Sección 5.1.2.

- **Reajuste de la red al conectar y desconectar nodos:** En esta implementación de la red los nodos se van conectando automáticamente entre ellos eligiendo según la potencia de señal de cada uno y los nodos que ya tienen conectados siguiendo la estructura de una red de malla. Sin embargo, como en este caso uno de los nodos necesita obligatoriamente de una red WiFi que sea la que proporciona la conexión con el *broker* MQTT, si los nodos detectan que esta red es lo suficientemente potente se conectarán a ella directamente. Por tanto, al desconectar un nodo de la red, además de perder los datos que estuviera recolectando, si éste tenía nodos conectados a él es necesario que estos se vuelvan a conectar a la red. Para volver a conectarse, el nodo en cuestión escaneará las redes WiFi disponibles que pertenezcan a la red de malla para la que está configurado y se conectará al dispositivo que cumpla las condiciones de diseño de la red. En este caso concreto, al estar todos los nodos conectados a la red WiFi del domicilio, al desconectar un nodo simplemente se pierde la conexión con él y al volverse a conectar se recupera.

- **Potencia de señal:** Para analizar la potencia de la señal WiFi en cada uno de los nodos durante el tiempo que han estado funcionando se ha utilizado la función WiFi.RSSI(), que proporciona la potencia de la señal en dBm. Como se ha explicado anteriormente, uno de los nodos está conectado a la red WiFi del *router* al que también está conectado el broker MQTT y el resto de nodos pueden conectarse también a esta red o a las redes formadas por cada uno del resto de nodos. En este caso en concreto todos los nodos entran dentro del haz de la señal del *router* WiFi, por lo que los datos obtenidos son la potencia de la señal recibida en el ESP8266 desde el *router*. La potencia media calculada con los datos obtenidos del intercambio de los mensajes se encuentra en la Tabla 5.7. Como se ha visto, la distribución es la que se muestra en la Figura 5.1, y el *router* se encuentra situado en el salón, por lo que es coherente que los nodos que se encuentran más cerca de él (CC y GG) hayan obtenido una mayor potencia de señal y los más alejados (HH y BB) menor.

Nodo	Potencia (dBm)
AA	-72,06451613
BB	-75,55645161
CC	-51,125
DD	-63,62575453
EE	-60,68145161
FF	-74,98991935
GG	-51,46169355
HH	-76,5766129

Tabla 5.7: *Potencia media de señal de cada nodo de la red.*

- **Mensajes perdidos:** Todos los mensajes que se han enviado tienen un identificador numérico que se incrementa en una unidad cada vez que se envía un nuevo mensaje, por lo que ordenando los mensajes por nodo y este identificador se puede ver si se han enviado y recibido todos los mensajes. Buscando los posibles saltos entre los números de mensaje se ha visto que no se ha producido ninguno. Todos los mensajes han sido enviados y recibidos por el nodo AA que ha monitorizado la red. No se ha perdido ningún mensaje en la transmisión, por lo que la red con topología *Mesh* y con protocolo MQTT es muy fiable.

- **Retardos:** En este caso se ha intentado conseguir el retardo de los mensajes desde que se envían hasta que se reciben utilizando un servidor NTP que sincronizara todos los relojes de los nodos de la red con la hora proporcionada por el servidor. Sin embargo, ha habido dificultades a la hora de recolectar los datos de esta implementación de la red a través de uno de los dispositivos ESP8266, ya que el servidor NTP no se ha sincronizado correctamente en todos los nodos y algunos tenían diferencias de un par de minutos. Por tanto, en este caso los retardos se han calculado viendo las diferencias de tiempo entre la emisión de mensajes y la recepción de los mismos en cada chip. La diferencia entre la transmisión debería ser de 8 segundos y lo mismo para la recepción. El tiempo entre mensajes siembre debe ser de 8 segundos. Entre la transmisión de los mensajes se ha obtenido un tiempo de 8 segundos, sin embargo en la recepción hay

unos pocos que son algo superiores, que están entre 9 o 10 segundos, pero la media de tiempo entre recepción de mensajes es de 8 segundos, por lo que puede considerarse que el *delay* entre el envío y la recepción de mensajes es despreciable.

- **Potencia consumida:** Para calcular la potencia consumida por las placas se ha realizado un circuito físico con un multímetro que obtenga la corriente y el voltaje suministrados a uno y a dos dispositivos ESP8266 conectados en paralelo durante la realización del experimento. Los circuitos resultantes de esta medición pueden verse en las Figuras 5.2, 5.3, 5.4 y 5.5, como se ha visto anteriormente. El voltímetro se coloca en paralelo con las placas y el valor de la tensión es de *4,9 V* para el caso de una única placa y para dos. El amperímetro debe colocarse en serie con los dispositivos y los datos obtenidos para las dos pruebas pueden verse en la Tabla 5.8. En esta tabla también puede verse el resultado del cálculo de la potencia absorbida por las placas con el protocolo MQTT. Se ha obtenido con la función $P=V*I$ y el resultado está en microWatios (μW), siendo de 65,66 μW para una única placa y de 118,09 μW para dos conectadas en paralelo.

	Corriente (uA)	**Potencia (μW)**
Una placa	13,4	65,66
Dos placas	24,1	118,09

Tabla 5.8: *Potencia consumida por uno y dos nodos de la red.*

Análisis de distancia

Con el objetivo de comprobar a cuanta distancia pueden colocarse los dispositivos ESP8266 se ha realizado una prueba específica en la que se han comprobado la distancia máxima del nodo que debe estar dentro de la red WiFi donde se encuentre el broker MQTT y la distancia máxima entre nodos fuera de esta red.

Primero, ha sido necesario calcular a cuánta distancia el nodo que debe estar conectado a la red WiFi donde se encuentre el broker pierde la conexión. Una vez conocida esta distancia

ese nodo se coloca ahí y se calcula a cuántos metros puede conectarse el siguiente nodo, que se conectará a la red generada por el anterior.

El nodo que se ha conectado con el *router* ha sido el nodo AA, y para conocer a qué distancia se encuentra el borde de la conexión WiFi generada por el teléfono móvil que la proporciona se ha dejando el *router* fijo y se ha buscado el punto donde se pierde dicha conexión. Así, se ha obtenido una distancia de 33 metros, que puede verse en la Figura 5.22. Cabe mencionar que el teléfono que hace de *router* y el broker MQTT se encuentran dentro de una casa, mientras que el nodo AA se encuentra en el exterior.

Una vez conocido este punto, se ha realizado la prueba de distancia entre los dos nodos AA y BB de la red. En este caso, el nodo AA genera una red WiFi a la que se conectará el nodo BB. Manteniendo fijos el nodo AA, el *broker* y el *router* se ha ido moviendo el nodo BB en línea recta hasta que se ha perdido la conexión con el mismo. A una distancia de aproximadamente 195 metros en línea recta se ha perdido la conexión, como se muestra en la Figura 5.23.

5.2. Topología estrella

5.2.1. Protocolo TCP

La topología estrella es aquella en la que los dispositivos están conectados a uno que actúa de centro y a través del cual se realizan las comunicaciones. Por tanto, para este experimento, es necesario que uno de los chips ESP8266 actúe como servidor y reciba todos los mensajes del resto de chips, que actuarán como clientes.

Análisis en interior

La librería que se ha utilizado para este caso ha sido la librería ESP8266WiFi[6], que utiliza el protocolo TCP para las comunicaciones. Para la realización del experimento, se ha partido de dos de los ejemplos incluidos en la librería. Uno de ellos el del servidor de la red y otro el de los clientes. El chip que actúa como servidor es el que recibirá los mensajes y

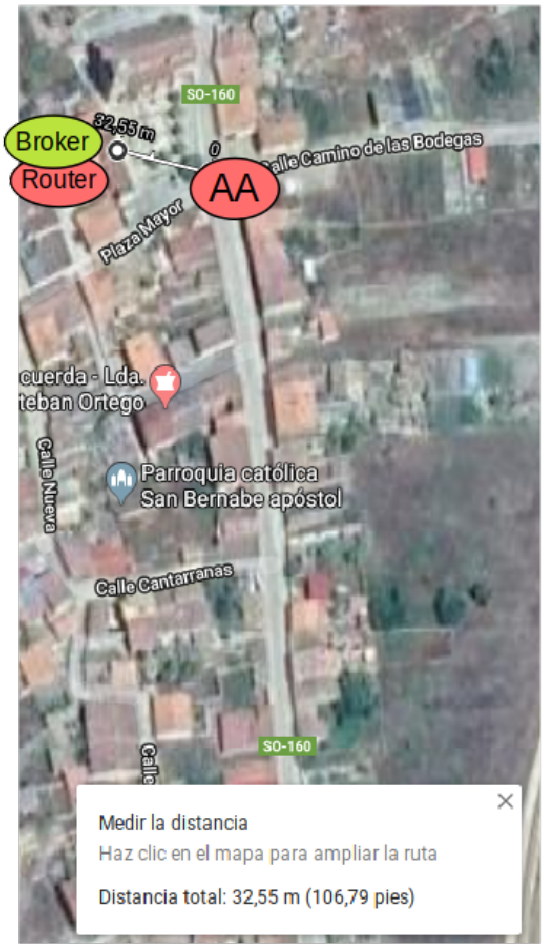

Figura 5.22: *Distancia entre el nodo AA conectado a la red WiFi y el router.*

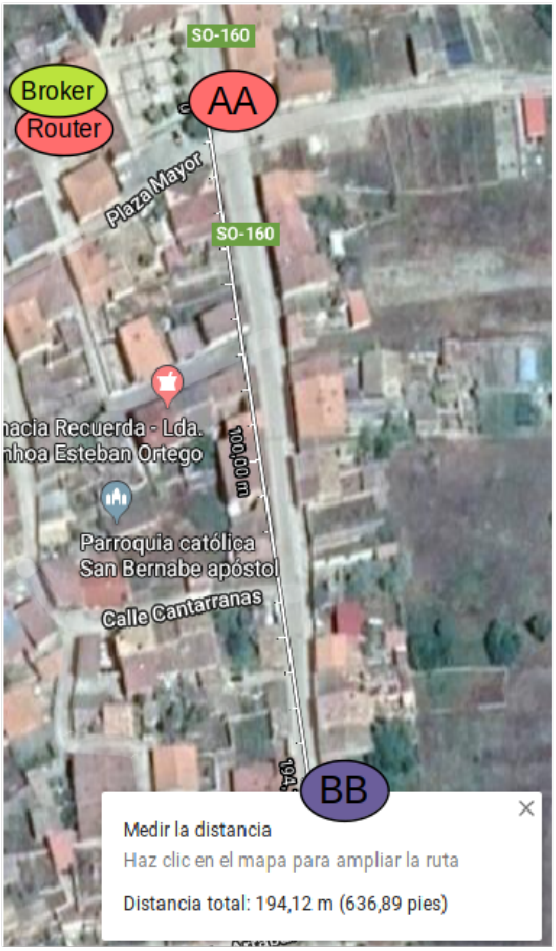

Figura 5.23: *Distancia entre los nodos AA y BB en una red mesh con MQTT.*

el que monitorizará la red a través del monitor serie y los que actúan como clientes estarán repartidos por el escenario y enviarán mensajes periódicamente al servidor. Todos los nodos deben estar conectados a la misma red WiFi, y los que actúen como clientes, además deben conectarse al servidor a través de la dirección IP que se le haya asignado en la red.

Ambos ejemplos se han modificado para obtener el estado de la red cada vez que se envía o recibe un mensaje. Los nodos que actúan como clientes envían mensajes con la estructura que se ve en la Tabla 5.9, en los que incluyen el nombre del nodo que envía el mensaje, un id del número de mensaje que se incrementa con cada envío, la potencia de señal WiFi y la hora a la que se envía el mismo. Los clientes emiten estos mensajes cada 8 segudos.

ID del nodo	número de mensaje	hora de envío	potencia señal emisor

Tabla 5.9: *Formato del mensaje enviado por los nodos cliente al servidor.*

Al recibir cada uno de los mensajes, el servidor añade su nombre, la hora de recepción y la potencia de señal WiFi que tiene en ese momento para mostrarlo por el monitor serie. Con estos datos, la apariencia final del mensaje que se muestra por pantalla es la que se ve en la Tabla 5.10.

ID del nodo receptor	potencia señal receptor	hora de recepción	
ID del nodo emisor	número de mensaje	hora de envío	potencia señal emisor

Tabla 5.10: *Formato del mensaje que el servidor muestra por el monitor serie.*

Para poder obtener la hora de envío y recepción de los mensajes ha sido necesario utilizar un cliente NTP que accediera a la hora suministrada por la red y que así estuvieran todos los nodos sincronizados. Para ello, se han añadido al programa las librerías NTPClient y WiFiUDP de Arduino que son necesarias.

Una vez se ha cargado el programa correspondiente al servidor en el nodo AA y el del cliente en el resto de nodos, se han distribuido los dispositivos por el espacio del domicilio siguiendo el esquema de la Figura 5.1.

El nodo AA que monitoriza la red debe recibir los mensajes que los clientes envían cada

ID receptor	Potencia receptor	Hora recepción	ID emisor	Nº mensaje emisor	Potencia emisor	Hora envío
AA	-76	13:13:12	BB	10	-77	13:13:12
AA	-76	13:13:15	HH	11	-67	13:13:14
AA	-76	13:13:19	CC	11	-42	13:13:19
AA	-76	13:13:20	FF	11	-66	13:13:20
AA	-75	13:13:20	GG	5	-64	13:13:19
AA	-75	13:13:20	DD	11	-63	13:13:19
AA	-75	13:13:20	EE	11	-73	13:13:20

Tabla 5.11: *Secuencia completa de estrella con TCP con mensajes de todos los nodos de la red.*

8 segundos con los datos descritos anteriormente, una secuencia completa de los mismos puede verse en la Tabla 5.11. El experimento ha recogido los mensajes de los nodos durante una hora y se ha analizado el comportamiento de la red.

- **Nodos reiniciados:** Ninguno de los nodos se ha reiniciado, todos han funcionado correctamente.
- **Nodos que han perdido la conexión:** Ninguno de los nodos ha perdido la conexión con el servidor.
- **Distancia:** Durante el tiempo que ha estado funcionando la red, ninguno de los nodos se ha visto afectado por la distancia.
- **Reajuste de la red al conectar y desconectar nodos:** Al desconectar cualquiera de los nodos cliente el servidor deja de recibir los mensajes de ese cliente, pero el resto de la red sigue funcionando sin problema. En caso de que el nodo desconectado sea el nodo que actúa como servidor, los nodos cliente pierden la conexión con el servidor y por tanto dejan de enviar mensajes. Hasta que no vuelva a conectarse el nodo servidor los clientes no mandarán mensajes, se quedarán esperando a que vuelva a aparecer en la red.
- **Potencia de la señal:** La potencia de señal WiFi la miden cada uno de los nodos

cada vez que envíen o reciben un mensaje. Está medida en dBm y la proporciona la propia librería ESP8266WiFi a través de la función WiFi.RSSI(). Esta potencia es la que le llega a cada nodo desde el *router* WiFi al que están conectados y los datos obtenidos pueden verse en la Tabla 5.12. Sabiendo que el *router* se encuentra en el salón y siguiendo la distribución del mapa de la Figura 5.1, es coherente con las cifras que el nodo que más potencia detecte sea el CC y el que menos el EE, ya que son el más cercano y el más alejado al *router* respectivamente.

Nodo	**Potencia (dBm)**
AA	-70,72004608
BB	-74,05893910
CC	-42,01171875
DD	-63,43469388
EE	-73,55876289
FF	-66,80457380
GG	-61,91848907
HH	-67,57520325

Tabla 5.12: *Potencia media de señal de cada nodo de la red.*

- **Mensajes perdidos:** Con el objetivo de estudiar la fiabilidad de la red en cuanto a número de paquetes perdidos entre el envío y la recepción de los mismos, se han estudiado los mensajes intercambiados durante el experimento. Para ello, cada mensaje envía un número identificador que aumenta en una unidad cada vez que se envía, por lo que basta con ver los paquetes que ha recibido el nodo que AA que actúa como servidor. Colocando los mensajes por nodo y número de mensaje enviado se buscan las anomalías en los números que deberían ser consecutivos. Se han encontrado 12 casos en los que el nodo AA ha recibido mensajes que no eran consecutivos. Se han detectado saltos como máximo de dos mensajes, lo que quiere decir que realmente sólo se ha perdido uno porque simplemente falta un número entre dos mensajes. El número total de paquetes perdidos para esta implementación es de 12. Es interesante mencionar que en todos los casos donde se detecta que falta un mensaje, el tiempo

entre el envío del paquete anterior y el siguiente es el equivalente al intervalo entre dos mensajes, por lo que puede que la pérdida del mensaje puede que esté en el envío o en la recepción del mismo.

- **Retardos:** Es interesante medir la diferencia de tiempos entre que un nodo envía un mensaje y el destinatario lo recibe. Los ESP8266 miden el tiempo transcurrido desde que se encienden, pero este parámetro no es útil para medir retardos entre dispositivos, ya que dependiendo de cuándo se haya encendido cada chip llevará una cuenta distinta. Por ello, se ha añadido un cliente NTP al programa que pueda acceder a la hora proporcionada por la red y así tener todos los nodos sincronizados. La librería utilizada es NTPClient, como se ha comentado anteriormente, que proporciona la hora con precisión de segundos. Esto dificulta el cálculo preciso del retardo, pero sirve para detectar anomalías durante el experimento. de los datos obtenidos, llama la atención que hay unos pocos paquetes que se reciben antes de ser enviados, algo que se debe a la poca precisión del servidor NTP. Por otra parte, se observa que la mayoría de los paquetes tienen *delays* de 0 ó 1 segundo, algo razonable. Haciendo la media de todos los retardos obtenidos durante el experimento se obtiene un valor de 2,55 microsegundos, un retardo es muy inferior a un segundo.

- **Potencia consumida:** La potencia consumida por los nodos durante el funcionamiento de la red se ha medido físicamente con un multímetro conectado a los pines de la alimentación del chip. Para calcularla se ha utilizado la fórmula $P=V*I$, por lo que se han obtenido la corriente absorbida por el dispositivo y el voltaje suministrado al mismo. Ya que se trata de medidas muy pequeñas, se ha probado a hacer la medida teniendo primero solo un chip conectado y después dos en paralelo. La colocación del multímetro actuando como amperímetro del circuito y uno de los dispositvos ESP8266 puede verse en la Figura 5.2. Las mediciones con dos placas conectadas simultáneamente se han hecho colocando la segunda placa en paralelo con la que ya hay, de tal forma que la corriente total sea la suma de las dos que van a las ramas de los chips.

Para obtener el voltaje, el multímetro ha funcionado como voltímetro y se ha colocado siguiendo el esquema de la Figura 5.5, ya que el voltaje es igual en cada una de las ramas del circuito. El voltaje obtenido es de *4,9 V* y los datos de las corrientes para una y dos placas, junto con los resultados de la potencia calculada pueden verse en la Tabla 5.13. La potencia consumida por una placa es de 64,19 μW y por dos 111,23 μW.

	Corriente (uA)	**Potencia (μW)**
Una placa	13,1	64,19
Dos placas	22,7	111,23

Tabla 5.13: *Potencia consumida por uno y dos nodos clientes de la red.*

Análisis de distancia

Como en los casos anteriores, se ha realizado también una prueba de distancia entre nodos de la red. Para este experimento en concreto basta con comprobar a cuánta distancia se pierde la conexión de cualquiera de los dispositivos con la red WiFi que interconecta los nodos clientes con el servidor.

La prueba para este caso se realiza con la red WiFi que proporciona un teléfono móvil, por lo que la distancia será algo menor que la que se obtendría con un *router* convencional.

El programa que tienen los dispositivos para esta prueba es el mismo que el que se ha utilizado en el experimento anterior, uno de ellos actúa de servidor de la red y el resto son clientes. La única modificación que se ha realizado es que envíen los mensajes cada menos tiempo y poder detectar así más rápidamente la distancia a la que se pierde la conexión.

Una vez realizada la prueba se ha obtenido que la distancia máxima entre el *router* WiFi y el dispositivo ESP8266 que haga de cliente es de 134 metros en línea recta y en un ambiente sin muchas interferencias. En la Figura 5.24 vemos la medida de esta distancia.

Al perder la conexión del nodo BB con la red simplemente se han dejado de recibir los mensajes que estaba enviando. Al volver a acercar un poco el nodo al *router* se ha vuelto a restablecer la conexión y se han vuelto a recibir los mensajes. Se ha observado que no

se ha perdido ningún paquete entre la pérdida y la recuperación de la conexión, ya que el programa detiene el envío de mensajes cuando el chip no está conectado a la red.

5.2.2. Protocolo MQTT

Para analizar el protocolo MQTT explicado en la Sección 4.1.2 con topología en forma de estrella, se ha utilizado el Framework de Arduino con la librería PubSubClient [8], que da soporte a una red de este tipo. El autor facilita distintos ejemplos de utilización de la librería, desde el caso básico de uso hasta la comunicación protegida con autenticación. En este caso, se ha partido del ejemplo básico proporcionado, pero se han añadido algunas modificaciones para poder recopilar datos sobre la potencia de la señal de los nodos o las horas de envío y recepción de los mensajes entre otros.

Debido al funcionamiento de MQTT, es necesario que los nodos de la red se conecten a un *broker* MQTT que se encargue de recoger y reenviar los mensajes de la red. Para este ejemplo se ha utilizado una Raspberry Pi Model B con el sistema operativo Raspbian Jessie y la librería *Mosquitto* instalada.

Análisis en interior

En esta prueba se han mantenido ocho nodos conectados simultáneamente durante una hora. Los dispositivos se han esparcido por un domicilio, de tal forma que tuvieran paredes y otros objetos que pudieran interferir en la transmisión de paquetes. La distribución inicial es la misma que para el resto de casos y que puede verse en el esquema de la Figura 5.1.

Como se ha indicado anteriormente, se ha utilizado el ejemplo de ESP8266 de la librería PubSubClient [8] de Arduino con algunas modificaciones. Además de configurar los parámetros de la red WiFi a la que están conectados los nodos y la dirección de la Raspberry Pi que actúa de Broker, se han modificado los *topics* de los mensajes de salida y entrada.

La primera comprobación que se hace es comprobar que los nodos reciban los mensajes de los demás con los dos *topics* de entrada y salida de mensajes. Para ello, se publica un mensaje en el *topic* "TFM/test/in"desde el cliente *Mosquitto* en la RaspberryPi y podemos

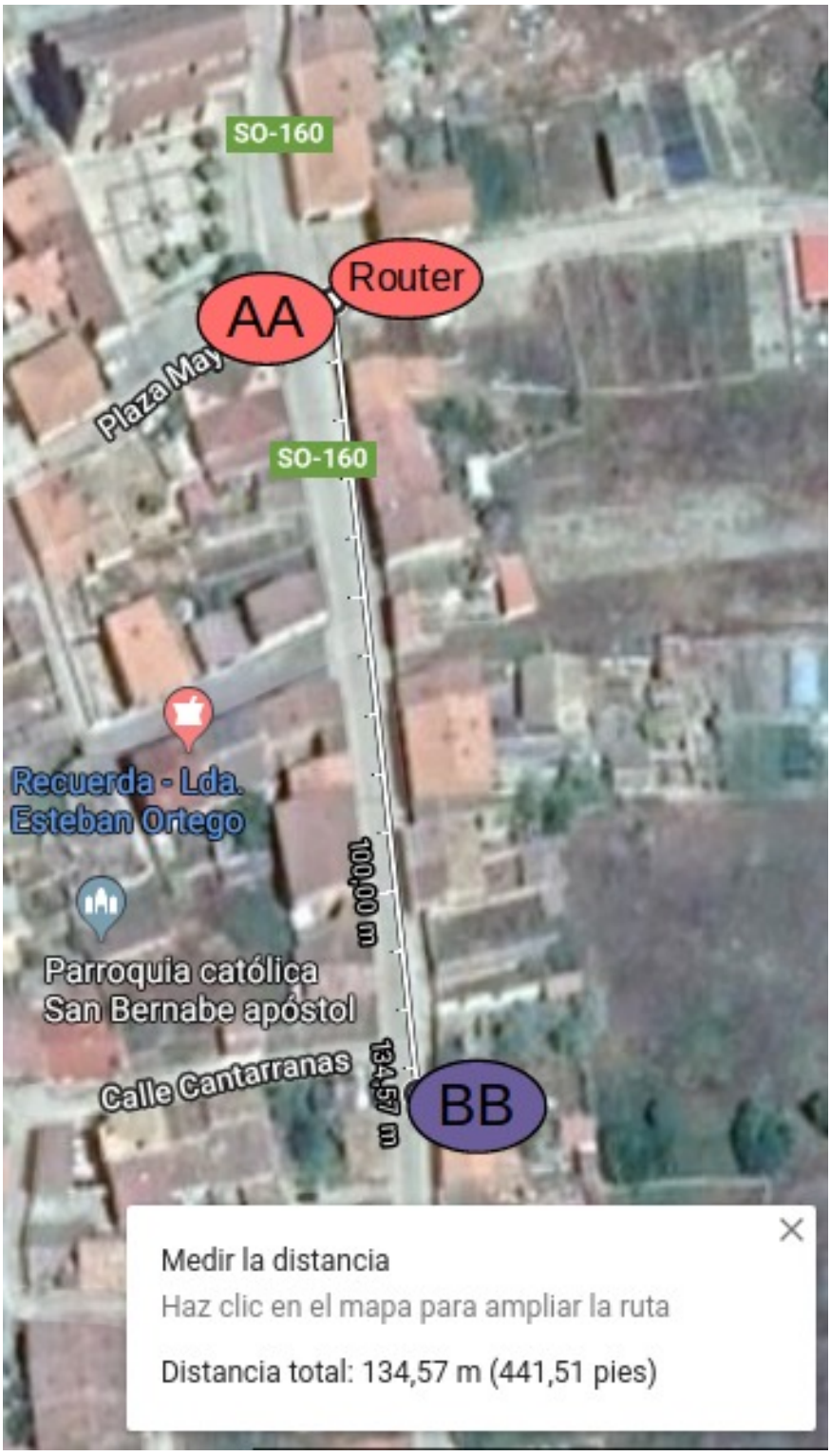

Figura 5.24: *Distancia entre el nodo AA que actúa de servidor y el BB como cliente.*

observar que el nodo AA, que está conectado al ordenador y dispone de monitor serie, lo recibe como se ve en la Figura 5.25. Ya que el resto de nodos de la red también están suscritos a este topic, también habrán recibido el mensaje. También se observa en la captura que el nodo está enviado mensajes al *topic* de salida, que este caso es "TFM/test/out".

```
Publish message: AA hello world #1321
Publish message: AA hello world #1322
AA: Message received [TFM/test/in] Prueba recibir mensaje
Publish message: AA hello world #1323
Publish message: AA hello world #1324
```

Figura 5.25: *Extracto del monitor serie del nodo AA en el que se ve que ha recibido el mensaje enviado desde la Raspberry Pi.*

Además, para diferenciar los mensajes se ha añadido una etiqueta a cada uno de los nodos que permita identificarlos. Con el objetivo de recopilar datos sobre el funcionamiento de la red se han utilizado también otras librerías:

- ESP8266WiFi es necesaria para crear el cliente de WiFi que utilizará la librería PubSubClient para el protocolo MQTT. Gestiona la conexión a la red y tiene la función RSSI que facilita la potencia de la señal del WiFi en cada uno de los nodos.
- NTPClient se utiliza para que la red pueda acceder a la hora proporcionada por internet y que los nodos tengan los tiempos sincronizados.
- WiFiUdp es utilizada por la librería NTPClient para poder acceder al servidor NTP que proporciona la hora real a la red.

El programa manda un mensaje cada ocho segundos con la estructura que se muestra en la Tabla 5.14.

ID del nodo	número de mensaje	hora de envío	potencia señal emisor

Tabla 5.14: *Formato del mensaje enviado por los nodos a la red.*

El número de mensaje se incrementa con cada mensaje enviado, y únicamente se reinicia la cuenta cuando el chip se apaga.

Al recibirlo, los dispositivos muestran por el monitor serie un mensaje que incluye los datos del momento de recibirlos y el texto recibido, como puede verse en la siguiente Tabla (5.15).

Topic MQTT	ID del nodo receptor	potencia señal receptor	hora de recepción
ID del nodo emisor	número de mensaje	hora de envío	potencia señal emisor

Tabla 5.15: *Formato del mensaje recibido por los nodos a la red.*

Una vez se ha cargado el programa en las ocho placas, se han repartido por el escenario y se ha observado el comportamiento de las mismas a través de la librería *Mosquitto* directamente en la RaspberryPi, con el programa MQTT.fx en el ordenador y la salida del monitor serie de uno de los dispositivos.

Para la realización del análisis se ha dejado a la red durante una hora intercambiar mensajes. Una vez recopilados los datos del monitor serie se han analizando utilizando una hoja de cálculo de *Google*. Un ejemplo de una secuencia completa con mensajes de todos los nodos de la red es el que puede verse en la Tabla 5.16. El nodo conectado al ordenador y desde el que se ve el monitor serie es el nodo AA, por lo que todos los mensajes serán recibidos por este.

La valoración de la red se detalla a continuación.

- **Nodos reiniciados:** Ninguno de los nodos ha perdido la conexión con el *broker*.
- **Nodos que han perdido la conexión:** Ningún nodo se ha perdido durante el periodo de prueba.
- **Distancia:** En este escenario la distancia no debería ser un problema para el funcionamento de la red.
- **Reajuste de la red al conectar y desconectar nodos:**

Topic MQTT	ID receptor	Potencia receptor	Hora recepción	ID emisor	Nº mensaje emisor	Potencia emisor	Hora envío
[TFM/test/1]	AA	-72	18:47:25	FF	13	-68	18:47:25
[TFM/test/1]	AA	-73	18:47:25	EE	13	-66	18:47:25
[TFM/test/1]	AA	-73	18:47:25	HH	23	-64	18:47:25
[TFM/test/1]	AA	-72	18:47:29	AA	11	-71	18:47:29
[TFM/test/1]	AA	-72	18:47:29	BB	13	-79	18:47:29
[TFM/test/1]	AA	-71	18:47:31	CC	4	-48	18:47:31
[TFM/test/1]	AA	-72	18:47:32	DD	10	-68	18:47:32
[TFM/test/1]	AA	-71	18:47:42	GG	1	-58	18:47:42

Tabla 5.16: *Secuencia completa con mensajes de todos los nodos de la red.*

  - Al desconectar cualquier nodo simplemente los mensajes que envían dejarán de llegar al *broker* y se perderán.
  - En caso de que se desconecte el broker los nodos pierden la conexión con el mismo y dejarán de enviar los mensajes.

- **Potencia de la señal:** La potencia de la señal la mide cada nodo cada vez que envía y recibe un mensaje. El dato es la intensidad de la señal de la red WiFi, por lo que cuanto más cerca esté el nodo del *router* más fuerte será. En la Tabla 5.17 se recogen los datos medios de la potencia de cada uno de los nodos durante el experimento realizado. El *router* se encuentra en la habitación más grande, cerca del nodo CC, por lo que observando el mapa de la Figura 5.1 tiene sentido que la potencia de los nodos AA y BB sean las más bajas, ya que son los que se encuentran más alejados del mismo. Como se ha comentado anteriormente, este dato lo facilita la biblioteca ESP8266WiFi y se consigue con la función RSSI, que devuelve la potencia de la señal en dBm.

- **Mensajes perdidos:** Cada uno de los nodos de la red al mandar el mensaje le añade un número que se incrementa con cada nuevo mensaje, por lo que para ver si hay mensajes perdidos basta con ver si han llegado todos los números consecutivos cada nodo. El nodo que más destaca por el número de paquetes perdidos es el nodo HH, ya que en dos ocasiones ha habido siete paquetes que el nodo AA no ha llegado a

Nodo	Potencia (dBm)
AA	-73,04470588
BB	-79,30483271
CC	-57,34459459
DD	-68,70487805
EE	-64,01530612
FF	-68,67401961
GG	-68,55477032
HH	-64,0719697

Tabla 5.17: *Potencia media de señal de cada nodo de la red.*

	Valor máximo	Total saltos
AA	2	4
BB	4	5
CC	5	7
DD	4	7
EE	5	12
FF	5	9
GG	5	6
HH	7	8

Tabla 5.18: *Mensajes perdidos por los nodos en estrella con MQTT.*

recibir. Además el tiempo de espera entre el último mensaje recibido y el siguiente es el mismo que si los siete mensajes perdidos se hubieran enviado, por lo que puede que el error haya estado en la recepción de los mensajes y no en el envío. Lo mismo les ha ocurrido a los nodos CC, EE, FF y GG pero con cinco mensajes, el nodo AA no los ha recibido aunque el tiempo transcurrido entre el último recibido y el siguiente sí es correcto. Al nodo BB le ha ocurrido también una vez con cuatro mensajes perdidos. En la Tabla 5.18 pueden verse estas cifras, pero hay que tener en cuenta que son los valores máximos de la diferencia entre los números de mensajes y la suma total de estos saltos diferentes de 1. La suma total de saltos detectados da una cifra de 58 mensajes pero el valor real de paquetes perdidos será mayor, ya que se han detectado saltos mayores de 2.

- **Retardos:** Para medir los retardos de los paquetes en la red ha sido necesario añadir

un servidor NTP para que la hora estuviera sincronizada en todos los nodos. El servidor NTP proporciona la fecha y la hora con precisión hasta los segundos, por lo que es complicado medir retardos muy precisos. En el caso de la red que se ha implementado no debería haber retardos de más de un segundo, ya que el tamaño de los mensajes es pequeño y tienen una conexión fuerte. Debido a la poca precisión del segundo para medir retardos, se han dado algunos casos en los que la hora de recepción del mensaje es menor que la de transmisión, lo que puede deberse al ajuste de los milisegundos. Por otra parte, se han detectado diferencias de unos segundos entre la transmisión y la recepción de algunos mensajes. La media total calculada de los retardos de los mensajes en la red durante el experimento ha sido de 3 milisegundos, muy inferior a un segundo.

- **Potencia consumida:** Para medir la potencia que consume un nodo de la red se ha utilizado un amperímetro que mida la corriente que absorbe el dispositivo durante el tiempo que esté encendido. Con la corriente y el voltaje suministrado a la placa se puede calcular la potencia consumida por las placas. Para obtener estos datos se han hecho dos pruebas distintas, una que mide el consumo de dos placas en paralelo y otra que mida lo que con sume sólo una de ellas. Los esquemas eléctricos para estos dos experimentos son los que se han comentado en el Capítulo 5, donde se encuentran las Figuras correspondientes. Al encender las placas se puede observar durante menos de un segundo en la pantalla del amperímetro que hay un pico de consumo y que inmediatamente se estabiliza y se queda en un valor constante tanto para las medidas tomadas con una placa y con dos. Sabiendo que el voltaje suministrado a las placas es constante de *4,9 V*, la potencia consumida se ha calculado con la fórmula $P=V*I$ y los datos obtenidos son los que pueden verse en la Tabla 5.19.

	Corriente (uA)	Potencia (μW)
Una placa	13,1	64,19
Dos placas	24,1	118,09

Tabla 5.19: *Potencia consumida por uno y dos nodos de la red.*

Análisis de distancia

Con el objetivo de comprobar el alcance de los sensores dentro de la red se ha hecho una prueba en exterior en la que se han utilizado dos dispositivos ESP-8266 conectados a la red WiFi de un teléfono móvil a la que también estaba conectado el broker MQTT.

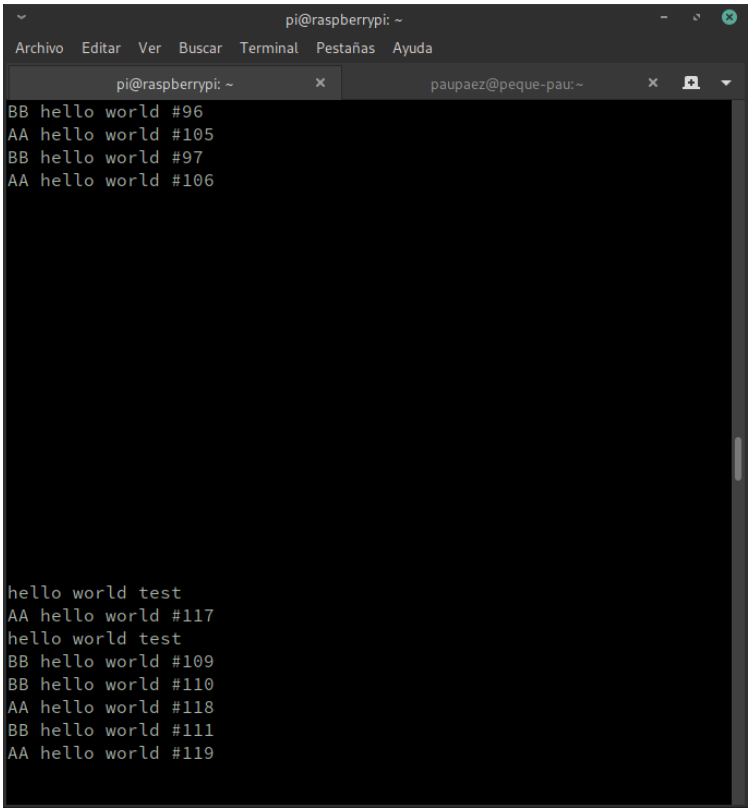

Figura 5.26: *Terminal de la Raspberry Pi durante la pérdida de conexión de los nodos de la red.*

Utilizando el mismo programa de prueba descrito en el apartado anterior, se ha obtenido que la conexión y el intercambio de mensajes de los dispositivos no presentaban ningún problema hasta que estos perdieron la conexión WiFi. Aunque hayan perdido la conexión,

los ESPs continúan enviando los mensajes que tenían programados, por lo que se pierde la información hasta que se recupera la conexión con la red. En la Figura 5.26 podemos ver los mensajes recibidos en la terminal MQTT de la Raspberry Pi antes de perder la conexión, al perderla y al recuperarla. Puede observarse que tanto el nodo AA como el BB han perdido 11 y 12 mensajes respectivamente en el tiempo que no han tenido conexión con la red y por tanto con el broker MQTT.

Los dispositivos pierden la señal del WiFi del teléfono móvil a unos 65 metros del mismo. La prueba se ha realizado en línea recta y en el exterior, como puede verse en el mapa de la Figura 5.27.

Hay que tener en cuenta que el alcance de las redes WiFi de los teléfonos móviles suele ser más bajo que las de los *routers* convencionales, por lo que según las necesidades que presente el sistema a implementar convendría tener en cuenta la distancia máxima de los nodos con el *router* de la red.

5.3. Resumen de los resultados obtenidos

En esta sección se han analizado los comportamientos de los chips ESP8266 en distintas configuraciones de redes con topologías *mesh* y estrella y protocolos TCP y MQTT. A continuación se recogen los datos obtenidos para cada uno de estos tipos de red, ordenados por cuadros.

Para la topología *mesh* se han obtenido los resultados que se recogen en la Tabla 5.20. Como puede verse en ella, ambas topologías han obtenido resultados bastante parecidos, no se pierden conexiones con los nodos, ni paquetes en la transmisión de los mensajes. Además, los retardos entre la emisión y la recepción de los mensajes es de media inferior a un segundo, los consumos de energía en ambos casos son similares y las distancias entre nodos han obtenido resultados parecidos. La única diferencia significativa en esta topología es el la forma de conexión entre los nodos, ya que MQTT necesita el *router* WiFi para el funcionamiento de la red. Esto se debe que la forma de creación de las redes de los protocolos MQTT y TCP

Figura 5.27: *Mapa con la distancia a la que se pierde la conexión de los nodos de la red con el router Wifi.*

es diferente, como se explica en el Capítulo 4. Con estos resultados parece difícil seleccionar una de las dos redes frente a la otra, ya que en términos de fiabilidad, retardos y consumo se han obtenido datos muy similares. Para el caso de la topología *mesh* sería conveniente

analizar las necesidades del proyecto en cuanto a protocolo e implementación de la red, ya que también depende de que sea viable o no incorporar la red WiFi con el *broker* MQTT.

	Mesh	
	TCP	**MQTT**
Nodos que han perdido la conexión	Ninguno	Ninguno
Potencia de señal	Se crean pequeñas redes WiFi de cada uno de los nodos. El sistema se conecta automáticamente al que más potencia tiene y cumple las condiciones del número de hijos.	Cada nodo genera una pequeña red WiFi. Es necesaria una red WiFi a la que esté conectada el broker MQTT. Si los nodos detectan esta red, se conectan a ella y si no, al nodo con más potencia y que cumpla las condiciones de número de hijos.
Mensajes perdidos	0	0
Retardos	<1 segundo	<1 segundo
Potencia consumida	62,72 μW	65,66 μW
Distancia máxima en el exterior	200 metros	nodo-router: 33 metros entre nodos: 195 metros

Tabla 5.20: *Cuadro resumen de los resultados de la topología mesh*

En cuanto a los resultados de la topología estrella, se recogen en la Tabla 5.21. Para este caso los resultados que se han obtenido en cuanto a calidad de transmisión de mensajes en la red son algo peores que para las redes *mesh*. En ninguno de los protocolos se han perdido las conexiones con los nodos, pero sí ha habido mensajes que no han llegado a su destinatario. En el caso de TCP ha habido 12 mensajes que no se han entregado y en el de MQTT más de 58. El retardo entre el envío y la recepción de los mensajes es inferior a un segundo en los dos casos y el consumo igual. La potencia de la señal al tratarse de la topología estrella, depende directamente de la distancia a la que se encuentren los nodos del *router*. La diferencia más significativa en esta topología se ha observado en la distancia máxima entre el *router* y el nodo, ya que en el caso de TCP se ha obtenido una distancia máxima de 134 metros, mientras que en MQTT la distancia se reduce a la mitad. A la vista

de estos resultados, el protocolo TCP en la topología estrella sería algo mejor que MQTT, ya que ha perdido un paquete menos durante la prueba y además la distancia máxima entre el *router* y los nodos también es superior a la obtenida en MQTT.

	Estrella	
	TCP	**MQTT**
Nodos que han perdido la conexión	Ninguno	Ninguno
Potencia de señal	Potencia de la señal del router WiFi captada por el nodo. Cuanta más distancia entre router y nodo, menor potencia.	Potencia de la señal del router WiFi captada por el nodo. Cuanta más distancia entre router y nodo, menor potencia.
Mensajes perdidos	12	>58
Retardos	<1 segundo	<1 segundo
Potencia consumida	64,19 μW	64,19 μW
Distancia máxima en el exterior	134 metros	65 metros

Tabla 5.21: *Cuadro resumen de los resultados de la topología estrella*

Con estos resultados, se puede concluir que cualquiera de las redes de la topología *mesh* es mejor que las de estrella. Dentro de esta, no se han obtenido diferencias muy significativas entre los protocolos analizados en cuanto a la fiabilidad de la red. La más llamativa es la diferencia entre las distancias máximas, ya que en el caso de MQTT es necesario que uno de los nodos se encuentre dentro de la red WiFi en la que está el *broker* MQTT, y esto puede ser un problema según las necesidades de la red a implementar. Sin embargo, es una condición que no es difícil que se cumpla en el caso de las redes de sensores o de IoT, por lo que para seleccionar uno de los protocolos es conveniente repasar las diferencias más significativas entre MQTT y TCP. A la vista de los resultados, la elección del protocolo a utilizar debería hacerse según las especificaciones del proyecto.

MQTT es un protocolo que funciona sobre TCP, como se comentó en la Sección 4.1.2, y es de los protocolos más utilizados en IoT[17]. Esto se debe a que presenta ciertas ventajas

frente a TCP y algunas de ellas son las siguientes:

- Es un protocolo de comunicaciones pensado para comunicaciones entre máquinas.
- Su funcionamiento se basa en la publicación/subscripción, por lo que es un protocolo que facilita la escalabilidad del proyecto, ya que los clientes son independientes.
- Es un protocolo sencillo y ligero, por lo que permite que se utilice en dispositivos con especificaciones limitadas en cuanto a memoria o capacidad de proceso, muy típicos en IoT.
- No necesita un ancho de banda muy grande, por lo que facilita la implementación de redes con medios inalámbricas o con poca calidad.
- Dispone de medidas de seguridad para las redes que lo implementan.
- Tiene parámetros de configuración para la calidad del servicio para ajustarse a las necesidades de cada proyecto.
- Es un protocolo que se desarrolló en 1999[1] que ha demostrado que es fiable y robusto.

Por estas razones, sería conveniente la utilización del protocolo MQTT frente a TCP en la topología *mesh* siempre que se cumplan las condiciones de distancia que se han comentado en esta sección.

[1] http://mqtt.org/faq

Capítulo 6

Escenario real de aplicación

Finalmente, para corroborar los resultados obtenidos en el análisis de las topologías y protocolos, se ha implementado un escenario de aplicación real en la vida cotidiana. Para ello, se ha desplegado una red de dispositivos ESP8266 conectados entre ellos y desplegados por un domicilio que potencialmente sirva para mejorar la vida de las personas. A estos dispositivos ESP se les conectarán distintos sensores y actuadores que ayudarán a los inquilinos a vivir más fácilmente.

Con este despliegue en concreto no se busca implementar un sistema domótico convencional que controle los electrodomésticos y aparatos electrónicos de un domicilio. Este proyecto pretende crear una serie de herramientas que faciliten la vida de personas mayores o con diversidad funcional o intelectual y de sus familiares o personas responsables. En muchas ocasiones estas personas necesitan irse a vivir a centros especializados o con familiares porque sus domicilios no están adaptados para que puedan realizar sus tareas cotidianas o para poder actuar con rapidez en caso de emergencia, por tanto es interesante la idea de que la tecnología pueda mejorar e innovar en este campo.

Por tanto, en este trabajo se propone realizar una red real con una de las redes descritas y analizadas en los Capítulos 4 y 5. Para elegir una de las cuatro, es necesario analizar qué necesidades tiene este proyecto en concreto y elegir así la que mejor se adapte en cuanto a funcionalidad dentro del domicilio y fiabilidad de la red.

Las características de las casas donde se desplegaría la red pueden variar dependiendo

del usuario que vaya a utilizarla, por ello el sistema debe funcionar correctamente en todos los domicilios. La característica más restrictiva de este proyecto es la fiabilidad de la red, ya que hay que asegurar que todos los mensajes de alerta se envían, se reciben y se procesan correctamente. Por tanto, la red que se utilice debe asegurar la llegada de todos los datos recopilados por los sensores. Por otra parte, los datos recopilados por la red se corresponden con los comportamientos de los inquilinos en el domicilio, por lo que hay que tener en cuenta la protección de los mismos para evitar posibles ataques o un uso malicioso.

El proyecto consta de dos partes en el despliegue y puesta en funcionamiento de la red. Por una parte es necesario analizar los datos recogidos por la red durante el tiempo que esté encendida para poder ajustar los niveles de los umbrales de actuación a los ritmos de vida del usuario. Por la otra parte está el propio sistema de gestionar las anomalías en los datos recolectados y avisar en caso de que sea necesario. En el primer despliegue de la red los valores de estos umbrales se asignan por defecto y según vaya pasando el tiempo se van adaptando.

El objetivo de la domotización de este tipo de casas no es simplemente que funcione autónomamente si no que el sistema detecte automáticamente posibles anomalías en el comportamiento del inquilino y poder actuar de manera acorde, ya sea avisando a familiares o actuando con los sistemas de seguridad correspondientes.

6.1. Implementación del prototipo del sistema

Dados los resultados obtenidos en los análisis de las distintas combinaciones de topologías y protocolos del Capítulo 5 y las necesidades del proyecto que se quiere implementar se ha seleccionado para este ejemplo una red *mesh* con protocolo MQTT de la Sección 5.1.2.

Viendo los resultados obtenidos en el estudio de las redes, las dos implementaciones de la topología de estrella han quedado descartadas debido a los malos resultados obtenidos respecto a paquetes perdidos y fiabilidad de la red. Sin embargo, entre las opciones de la topología *mesh* podría utilizarse cualquiera de las dos ya que los resultados son similares.

Las causas de que se haya seleccionado el protocolo MQTT frente a TCP son entre otras, que es un protocolo pensado para la comunicación entre máquinas, como es el caso del proyecto, puede seleccionarse la calidad de servicio en cuanto a la entrega de mensajes, y al tratarse de mensajes pequeños hay menos datos que procesar.

Para el desarrollo se ha utilizado una Raspberry Pi que actúe como *broker* MQTT y los distintos ESP8266 que llevarán conectados distintos sensores que se encargarán de captar los datos del entorno y enviar lo que sea necesario. En este caso no es necesario que haya un ordenador conectado por puerto serie a uno de los chips, ya que estos se conectarán directamente al *broker* que es el que se encargará de gestionar si se envían o no alertas. Los nodos con sensores que componen la red envían mensajes distintos según si están detectando un dato dentro de los umbrales normales de comportamiento o no. Para que el usuario pueda ayudar al funcionamiento de la red y decidir si mandar o no mensajes de aviso al exterior de la red, se ha añadido un módulo que pide su confirmación para enviarlos al detectar un dato anómalo. Para este primer prototipo se han implementado 5 módulos con distintos sensores que envían la información recolectada a la red local del domicilio como los siguientes:

- **Sensor de fuego en la cocina:** módulo que pueda avisar de fuegos provocados por olvidos mientras se cocina.
- **Sensor de temperatura y humedad en el cuarto de baño, cocina o sala de estar:** módulo que detecte posibles caídas en el servicio o grifos que se queden abiertos. Además este sensor se utiliza para que analice la temperatura a la que suele estar el domicilio y que detecte si hay algo que se sale de la norma.
- **Sensores de luminosidad que detecten qué bombillas de la casa se encienden:** módulos que permitan sacar tanto un patrón de comportamiento del usuario en cuanto a los movimientos por la casa como detectar los comportamientos fuera de éste.
- **Sensor infrarrojo que detecte la apertura de puertas y ventanas:** módulo que recopila los datos de aperturas de puertas a lo largo del día y que detecta si hay

movimientos fuera de lo común.

- ***Wearable* para el usuario:** un módulo que se coloque el usuario a modo de pulsera o collar y que sirva para la confirmación de envío de mensajes de alerta. Es importante tener en cuenta que puede haber comportamientos detectados por sistema que no sean preocupantes y que no necesiten mandar mensajes de alerta, por lo que la red avisaría a la pulsera antes de enviar la alerta para que el usuario pueda decidir si enviarla o no.

Estos son los ejemplos de módulos que se han implementado para esta prueba en concreto, pero al tratarse de una red *mesh* MQTT de dispositivos ESP8266 podrían implementarse todos los módulos que hicieran falta con los mismos o distintos sensores y conectarlos a la misma malla. Los módulos que componen el sistema pueden alimentarse tanto con baterías o por corriente, aunque estando en un domicilio será más sencillo conectarlos a la red eléctrica directamente, exceptuando el módulo que llevará el usuario que sí debe alimentarse por baterías.

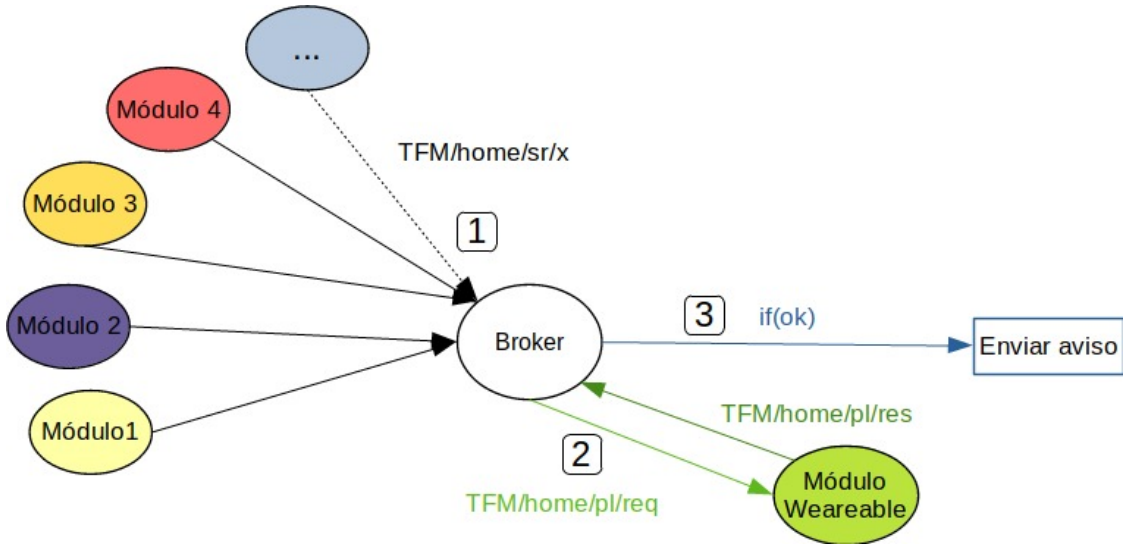

Figura 6.1: *Diagrama de funcionamiento de la red implementada.*

Los módulos mencionados envían mensajes con una estructura determinada y a un *topic* MQTT que se procesan en el *broker* con NodeRED. A la hora de implementar el sistema

	Habitación	Módulo	Dato	Hora
ALERT	Habitación	Módulo	Dato	Hora

Tabla 6.1: *Estructura de los mensajes de la red prototipo.*

se diferencia entre dos tipos de módulos, los que incorporan distintos sensores y el módulo *wearable*. Los primeros envían todos sus mensajes al *topic "TFM/home/sr/x"*, indicando en el campo x el sensor del que se toma el dato, cuando sea necesario y en función del número de mensajes por tipo que lleguen, el *broker* se encarga de enviar o no un mensaje al *wearable* pulsera con el *topic "TFM/home/pl/req"* que pida confirmación del envío del mensaje de alerta. La pulsera en ese momento mostrará por pantalla el texto correspondiente al tipo de alerta que esté recibiendo y en función del botón que pulse el usuario, enviará un mensaje al *broker* con *topic "TFM/home/pl/res"* confirmando que sí hay alerta o no, caso en el que no se realizará ninguna acción. El funcionamiento de la red puede verse de forma gráfica en el diagrama de la Figura 6.1.

Como se ha mencionado en el Capítulo 6, este proyecto consta de dos partes, la de la detección y envío de alertas y la del análisis de comportamiento del usuario para mejorar la precisión del sistema a la hora de detectar las anomalías. Por ello, los módulos de sensores envían dos tipos de mensajes: mensajes de recolección de datos periódicos y mensajes de detección de algo fuera de lo común. Los mensajes que recopilan los datos simplemente se almacenarán para procesarlos más adelante, y los de alerta son los que se utilizarán inmediatamente en la Raspberry Pi para que el sistema funcione como en la Figura 6.1. Con este objetivo, el cuerpo de los mensajes enviados en la red tienen una estructura fija determinada, que permita tanto el análisis como la gestión de alertas. En la primera fila de la Tabla 6.1 puede verse la estructura de los mensajes de análisis y en la segunda fila de la misma, la de los mensajes de alerta.

El código implementado para el funcionamiento de este prototipo se encuentra dentro de la carpeta *escenario* del repositorio de Github del apéndice B.

6.1.1. Broker MQTT

El *broker* MQTT es una de las partes más importantes del proyecto, ya que sin él la red dejaría de procesar los mensajes que envían los nodos. En este caso será, como ya se ha comentado, una Raspberry Pi conectada a una red WiFi.

Para ello implementarlo, se han instalado las librerías *mosquitto* de MQTT necesarias para que funcione y se ha utilizado el programa NodeRED[1] para gestionar los mensajes recibidos. NodeRED es una herramienta de IBM que permite la implementación de sistemas que necesiten comunicar *hardware* y *software* de forma visual y con todas las funcionalidades de las tecnologías que utiliza. [2]. Está diseñada para sistemas IoT, por lo que dispone de un amplio abanico de recursos a utilizar, entre los que se encuentran los protocolos MQTT y TCP utilizados en este trabajo. Además, se le pueden instalar paquetes adicionales para ampliar las utilidades del programa, como clientes de correo o redes sociales, muy útiles para programar *bots* que publiquen mensajes. Al ser una herramienta de código abierto, se pueden implementar nuevas funcionalidades si se necesitan. Las redes implementadas en NodeRED se crean a partir de nodos de diferentes tipos que permiten realizar tareas concretas y conectarse entre ellos para implementar el flujo del programa. Además, tiene unos módulos de funciones personalizables que se programan en *Javascript* que facilitan la implementación de las redes.

En este proyecto se ha desarrollado un flujo en NodeRED que pone a escuchar todos los mensajes que le lleguen por MQTT a uno de los *topics* mencionados en el apartado anterior: *TFM/home/sr/x* siendo x el identificador del sensor en cuestión o *TFM/home/pl/* para el módulo *wearable*. Al ser dos *topics* distintos se utilizan dos nodos de entrada de MQTT y dos flujos distintos, aunque hay algunos nodos que se comparten, como puede verse en la captura del flujo de la Figura 6.2.

Si se recibe un mensaje por el *topic* de los sensores se filtra el mensaje para detectar

[1] https://nodered.org/
[2] https://programarfacil.com/blog/raspberry-pi/introduccion-node-red-raspberry-pi/

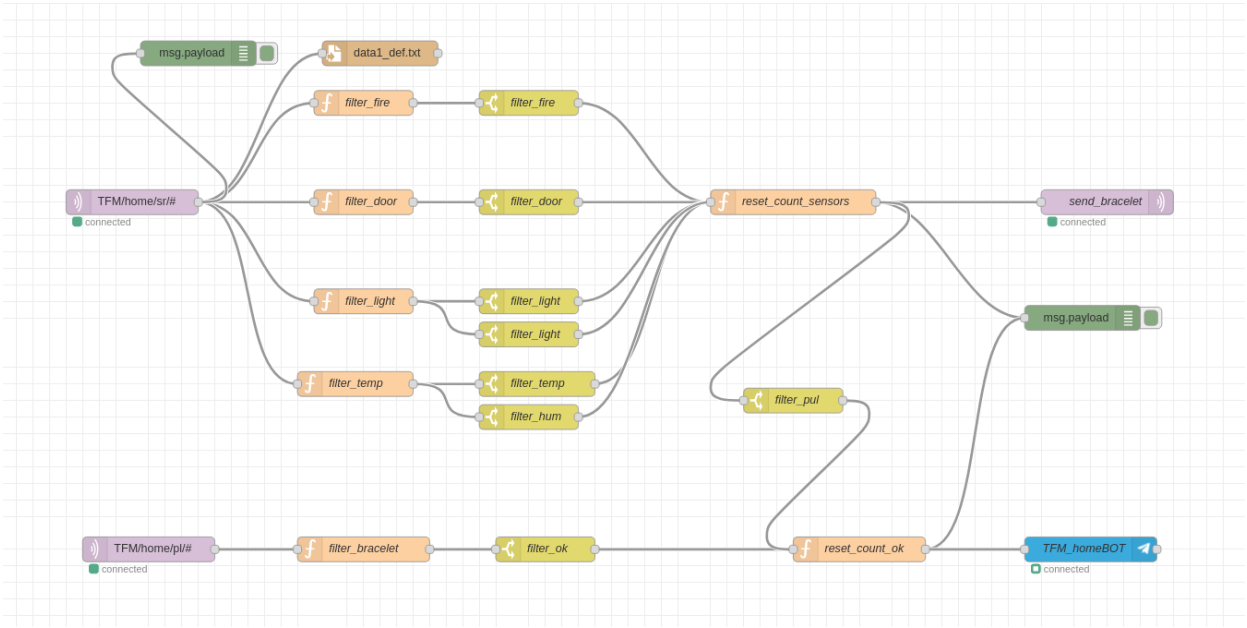

Figura 6.2: *Flujo del programa implementado para el prototipo en NodeRED.*

de cuál de los dipositivos se trata en la primera columna de nodos de color naranja con nombre *filter_xxxx*. Para cada uno de los tipos de sensor se inicia un contador que espera a tener un número determinado de cada uno antes de enviar el mensaje a la pulsera. Este segundo filtro se hace en la columna de nodos de color verde, y en el siguiente bloque se reinicia el contador que haya pasado su límite. A continuación se envía el mensaje MQTT correspondiente a este contador al *topic TFM/home/pl/in/*, que recibirá la pulsera.

En el bloque de reseteo de los contadores también se inicia una cuenta del número de mensajes que se envían a la pulsera, por si al enviar un número determinado de mensajes no se obtiene respuesta y es necesario enviar el aviso directamente. Actualmente está en 3 mensajes sin responder, pero pude ajustarse según las necesidades que se observen en el uso de la red. El funcionamiento del flujo correspondiente al segundo *topic* es igual que el descrito para el primero, pero configurando en el filtro naranja de la primera columna el mensaje final que se enviará al familiar en función de los recibidos por el sistema.

Para el sistema de alertas de este prototipo se ha instalado un paquete adicional de

NodeRED que permite enviar y recibir mensajes a través de la aplicación de mensajería Telegram. Así, cuando el usuario decide enviar el aviso o el sistema detecta que ha habido varios mensajes de alerta sin confirmar se enviará un mensaje a los usuarios de Telegram que se configuren en el último nodo del flujo. Para ello ha sido necesario crear un *bot* de Telegram que pueda gestionar la automatización de envío de mensajes.

Por otra parte, de cara a poder analizar los datos que se recogen en el *broker*, se ha añadido un nodo en el flujo de programa que permite almacenar todos los mensajes recibidos por el *topic* MQTT correspondiente a los sensores en un fichero de texto. Así, pueden estudiarse más adelante qué comportamientos son o no extraños y poder mejorar la precisión del sistema a la hora de mandar las alertas. Este fichero de texto en el que se guardan los datos podría enviarse diariamente con los datos recogidos a una centralita, o simplemente tener esta centralita conectada al mismo *topic* MQTT para recibir, almacenar y analizar los mismos datos.

Uno de los problemas de analizar los datos en una centralita es el tratamiento de los datos personales, por lo que sería recomendable hacerlo con ellos anonimizados para sacar normas generales de comportamiento. Con el objetivo de concretar más en el comportamiento específico de cada usuario los datos deberían procesarse en el mismo *broker*, que al ser una Raspberry Pi tiene capacidad para gestionar el análisis de los datos por semanas o meses.

6.1.2. Módulos de sensores

En esta sección se describen los distintos módulos implementados para el funcionamiento del prototipo descrito anteriormente.

Detector de fuego

Este módulo pretende detectar si en una casa se produce un fuego para avisar de inmediato a los sistemas de emergencia y actuar en consecuencia. Para su implementación se ha utilizado un sensor de fuego, un ESP8266 y un zumbador. El sensor de fuego es un sensor

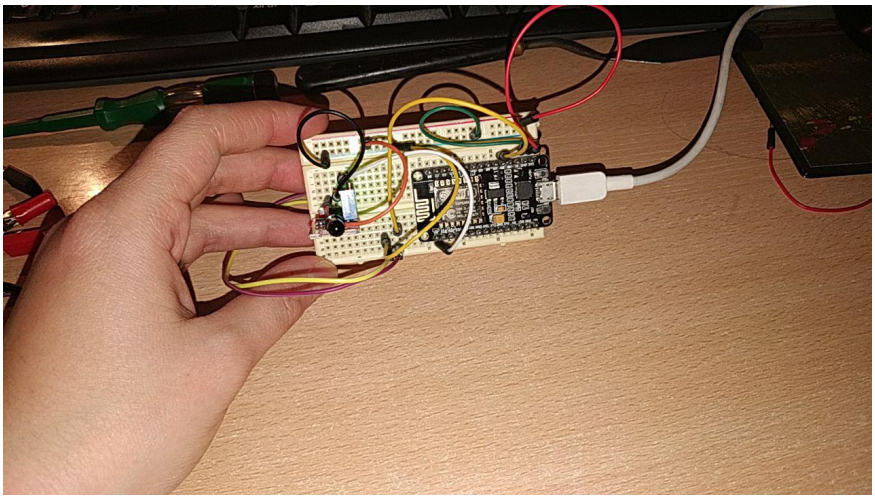

Figura 6.3: *Prototipo del módulo del detector de fuego.*

óptico que detecta la existencia de combustión por la luz que emite cuando se produce, ya que las longitudes de onda varían[3].

El prototipo en placa de desarrollo de este módulo puede verse en la Figura 6.3.

El programa implementado con el *framework* de Arduino inicializa el sensor y el zumbador y simplemente capta el valor que devuelva la lectura analógica del pin al que se ha conectado el sensor de llama. Cuando el valor leído sea menor que una cifra preconfigurada, se enviará un mensaje con la estructura de alerta al *topic* MQTT correspondiente.

Sensor de temperatura y humedad

Un sensor de temperatura y humedad puede ser muy útil para detectar si por ejemplo se quedan grifos o ventanas abiertos o comportamientos extraños en la temperatura de la casa. Este tipo de sensor también se puede aprovechar para gestionar la climatización del domicilio automáticamente, en función de la temperatura que se detecte y sin necesidad de estar pendiente[9].

Para este módulo se ha utilizado un ESP8266 y un sensor DHT11 de temperatura y humedad, que pueden verse en la Figura 6.4.

El programa que se ha implementado para este ESP8266 configura el sensor y lee los

[3] https://www.luisllamas.es/detector-llama-arduino/

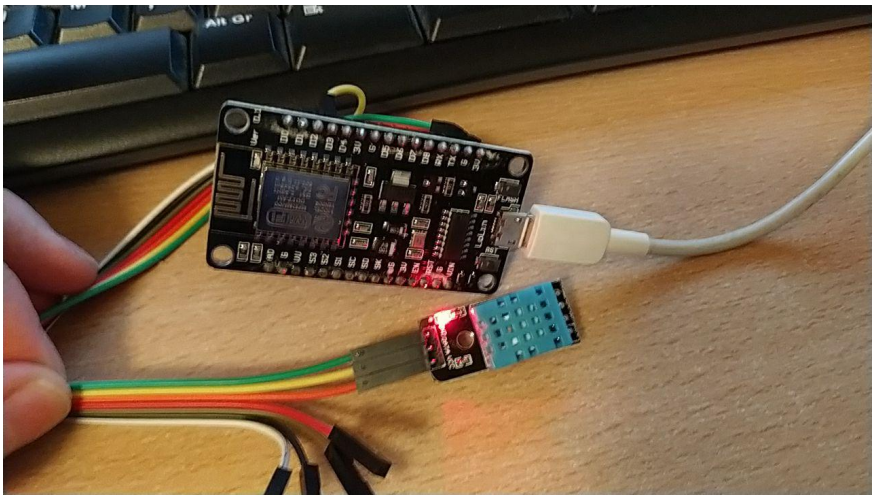

Figura 6.4: *Prototipo del módulo del sensor de temperatura y humedad.*

valores que le proporcionan periódicamente. Para leer los valores se ha utilizado la librería DHT de Adafruit[4], que realiza las conversiones a grados celsius automáticamente. En caso de que se reciban temperaturas o humedades fuera del rango establecido como normal se enviarán mensajes de alerta, y si no se enviarán mensajes de simple recogida de datos periódica.

Sensor de luminosidad

Podrían utilizarse sensores de luz tanto para alertar como para controlar luces o persianas del domicilio. De este tipo de módulo pueden colocarse varios iguales distribuidos para gestionar distintas habitaciones. Uno de ellos sería bueno instalarlo en el cuarto de baño, ya que es una habitación donde se entra y se sale y donde la luz se tiene encendida únicamente el tiempo que se está dentro. También sería útil en el dormitorio para detectar anomalías en las horas del sueño. Este módulo simple constaría de una placa ESP8266 y un fotorresistor que será el que capta la luminosidad del entorno y pueden verse en la Figura 6.5.

Este módulo se ha programado de tal forma que el dispositivo ESP inicialice los sensores y cuando lea un valor mayor o menor al umbral determinado se enviará el dato a través de MQTT. En este caso las alertas deben enviarse en función de las veces que se encienda una

[4] https://github.com/adafruit/DHT-sensor-library

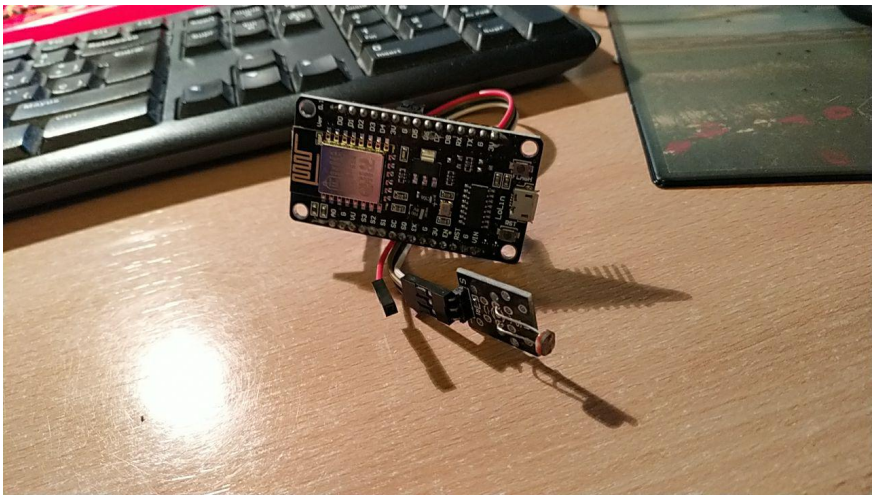

Figura 6.5: *Prototipo del módulo del sensor de luminosidad.*

bombilla, las horas a las que lo haga o el tiempo que esté encendida o apagada, por lo que en este módulo es importante ajustar bien los umbrales a la vida del usuario.

En el caso de que este módulo se utilice con el fin de controlar bombillas o persianas habría que colocarlos cerca de las ventanas de las habitaciones que se quieran controlar y poner los actuadores correspondientes. En el caso de las luces el actuador podría ser una conexión directa con las bombillas o un relé que se active cuando la cantidad de luz baje de determinada cifra y para las persianas podría utilizarse un controlador de motores que se active y desactive con las órdenes del ESP8266, aunque este caso es algo más complicado.

Detector de apertura de puertas o ventanas

Con este módulo se monitorizarán las puertas y ventanas que haya en el domicilio para detectar posibles anomalías en el comportamiento. Habría que estudiar según el usuario en qué puertas o ventanas sería útil colocar este sensor. Para implementarlo se ha utilizado un sensor infrarrojo que detecta la presencia o no de un objeto cercano y un ESP8266, como puede verse en la Figura 6.6. Este módulo se podría hacer también con sensores magnéticos que tengan un imán en la parte móvil de la puerta y el sensor en el marco de la misma.

Para la programación de este módulo se ha seguido el esquema de los tres anteriores.

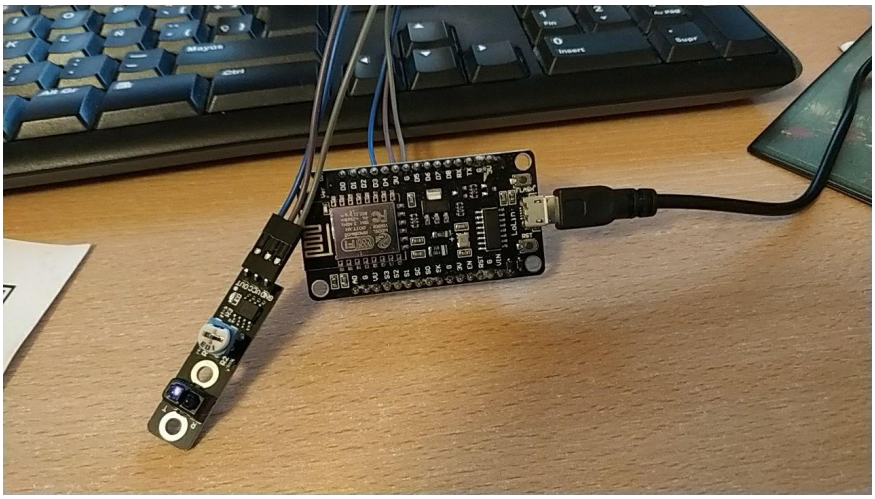

Figura 6.6: *Prototipo del módulo del detector de apertura de puertas y ventanas.*

Después de inicializar el sistema, el chip se queda escuchando si el sensor detecta cambios o no y en caso de que se abra o cierre una puerta se notifica. Igual que en el caso del sensor de luminosidad, en este es importante tener en cuenta los tiempos que permanece la puerta abierta o cerrada y la coherencia de los mismos con las costumbres del usuario.

Wearable para el usuario

Este módulo es un dispositivo que llevará el usuario en la muñeca o colgado del cuello y servirá para confirmar el envío de los mensajes de alerta. Para este módulo se ha utilizado un ESP8266 con una pantalla incorporada que se utiliza para mostrar los mensajes al usuario y dos botones, uno de confirmación de envío y otro de rechazo. Concretamente, la placa es una Heltec WiFi Kit 8 como la que se ve en el prototipo del módulo de la Figura 6.7.

El desarrollo del programa que lleva este módulo es algo más complejo que los anteriores, ya que tiene que gestionar la recepción de mensajes a través del *topic TFM/home/pl/in/* y la gestión de los mismos para mostrar los mensajes por la pantalla. Para la utilización de la pantalla del chip se ha utilizado la librería U8g2lib de Arduino[5]. Para la recepción de mensajes se ha configurado una función *callback* de la red que cuando se detecte algún mensaje con el *topic* configurado como entrada se ejecute el análisis del mensaje recibido

[5] https://github.com/olikraus/u8g2

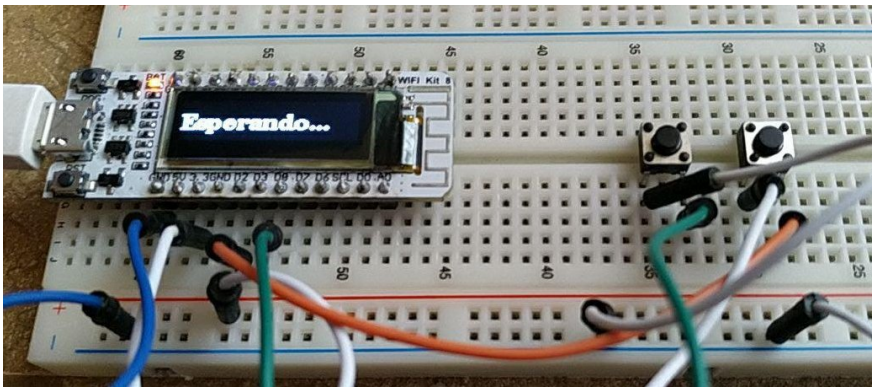

Figura 6.7: *Placa Heltec WiFi Kit 8.*

para mostrar por pantalla de qué alerta se está pidiendo la confirmación para el envío de la misma fuera de la red. En el bucle principal del programa es donde se gestionan las interrupciones de los botones en caso de que se haya recibido previamente una alerta. Si se pulsa el botón de OK se envía un mensaje de vuelta a la red al *topic TFM/home/pl/out/* que gestionará el *broker*. En caso de pulsar el botón CANCEL, se anula el envío de la alerta.

La idea este módulo es que vaya encapsulado en plástico o algún material resistente que proteja y fije los componentes con el objetivo de que sea un dispositivo cómodo y funcional similar a las pulseras inteligentes. La funda debe llevar espacio también para la fuente de alimentación que en este caso sí debe ser portátil y con autonomía suficiente para varios días.

De este tipo de módulo en caso de que el sistema llegara a comercializarse se deberían implementar distintos tipos que se adapten a las necesidades de todos los posibles usuarios, ya que puede que haya personas que puedan leer los mensajes en pantallas tan pequeñas. Lo ideal sería tener un modelo con asistente de voz para personas con discapacidad visual, pero en este prototipo solo se ha implementado el modelo con pantalla.

Capítulo 7

Conclusiones

7.1. Conclusiones e implicaciones del proyecto

En este trabajo se han estudiado y analizado algunas implementaciones de redes IoT de dispositivos ESP8266. Para ello se han desarrollado programas de prueba para los chips que han permitido la recolección de datos durante el funcionamiento de cada tipo de red para poder analizar y concluir algunos datos sobre su comportamiento. Una vez se han obtenido y estudiado lo datos de todas las redes, se ha implementado un sistema real de aplicación de una de ellas. Este escenario real de aplicación ha sido una casa domótica con sistema de alertas para la detección de anomalías en el comportamiento del inquilino de la casa. El objetivo de las alertas es poder avisar a familiares o servicios responsables para que se pueda recibir ayuda a tiempo en caso de que sea necesario. Para la realización del primer prototipo de este sistema ha sido necesario plantear qué funcionalidades debía tener el sistema, qué módulos debe incluir y cómo implementarlo.

El objetivo principal especificado al inicio de este documento era el análisis de distintos tipos de redes de ESP8266. Como se ha mostrado a lo largo de este documento, y concretamente en el Capítulo 4, se ha realizado la comparativa entre 4 redes con topologías y protocolos combinados 2 a 2. Las redes analizadas finalmente han sido redes con topología *mesh* y protocolos MQTT y TCP y redes con topología estrella y protocolos MQTT y TCP. Los resultados obtenidos han mostrado que la topología de estrella con cualquiera de los

protocolos da peor calidad en cuanto a la fiabilidad de la red que la topología *mesh*. En *mesh* las diferencias entre las prestaciones de los protocolos no son muy llamativas, por lo que conviene seleccionar uno de los dos teniendo en cuenta la aplicación final. Aunque este análisis puede ampliarse a multitud de combinaciones de topologías y protocolos, se considera que el objetivo principal ha sido alcanzado, ya que se han seleccionado dos tipos de redes representativas. La topología *mesh* y el protocolo MQTT son tecnologías muy utilizadas en sistemas IoT y la topología estrella y el protocolo TCP son más convencionales dentro del mundo de internet.

Otro de los objetivos planteados al inicio era el de la implementación de un sistema real con una de las tecnologías utilizadas. En el Capítulo 6 de este documento se describe la implementación del mismo, por lo que también se considera que este objetivo se ha alcanzado. El sistema se ha implementado con topología *mesh* y protocolo MQTT, ya que analizando las necesidades del proyecto se ajustaba más a ellas. MQTT está pensado para la comunicación entre máquinas y además permite seleccionar la calidad del servicio en el envío de mensajes, algo conveniente para la aplicación. Por otra parte, el desarrollo de este tipo de sistemas puede hacerse mucho más complejo y con más número de módulos de sensores y funcionalidades. Al tratarse de una red *mesh* con protocolo MQTT de tipo modular es escalable y se deja abierta la posibilidad de implementar y añadir nuevos módulos al sistema sin mucha dificultad.

A nivel personal, la realización de este proyecto ha implicado la adquisición de nuevos conocimientos sobre redes de IoT (o Internet de las cosas) y la forma de implementarlas. Con el análisis de los distintos tipos de redes he aprendido qué parámetros eran necesarios para la comparativa y cómo conseguirlos. Por la parte de la implementación del sistema real he aprendido a enfrentar un diseño desde cero de un sistema de redes real, desde idearlo hasta implementarlo.

7.2. Líneas futuras

Este proyecto se plantea como la introducción a un análisis de diferentes redes de dispositivos ESP8266, por lo que la línea futura más inmediata debería ser la comparativa entre más tipos de redes, topologías y protocolos durante su funcionamiento.

También hay posibles mejoras en los análisis realizados, como pueden ser mejorar la precisión de los tiempos de retardo entre el envío y recepción de los mensajes de la red o incluir nuevos parámetros que permitan determinar más características de la misma.

El prototipo de sistema real de aplicación también tiene muchas cosas que mejorar en caso de que se quisiera explorar un producto viable de mercado. Algunas de las cosas a añadir serían más módulos de sensores y el análisis en tiempo real de los datos obtenidos para el ajuste y personalización del sistema de alertas.

Capítulo 8

Introduction (English)

8.1. Motivation

Nowadays we live in a world where the development and the presence of technology in our lives increase by the moment. The goal of communicating between different parts of the planet through technological devices in real time has resulted in a hyperconnected world in which practically anywhere you have access to the largest data network, the Internet.

Since the first computers connected to the Internet, the evolution of technology has grown exponentially. Today, many of the devices used in daily life have integrated programmable chips that can be connected to the Internet in order to make them even easier to use. This expansion of Internet connections into objects of everyday use beyond computers was named the Internet of Things or IoT for the first time in 1999[1].

Today the Internet of things is widespread at all levels, from industry or agriculture to everyday life. As a result, a multitude of technologies and devices have emerged that aim to make IOT users understand how it works. These electronic components with didactic objectives are very varied and are adapted to the needs of many projects, such as the Arduino or RaspberryPi development boards. The existence of this devices and the interest in communication between them have led to the existence of groups of developers of new functionalities for the chips, as well as the generation of new project ideas.

[1] http://www.bcendon.com/el-origen-del-iot/

This work seeks to analyse some of these new technologies available to the majority of the population in relation to Internet of Things networks. This type of devices are accessible to users without much technical knowledge due to the extensive documentation on them and their low cost. The expansion of IoT has also caused these chips to include Internet connections and libraries that allow networks with their own structures to be implemented. Therefore, some of these functionalities will be analysed and the operation of these networks will be checked in real cases to obtain data on their behaviour.

Although this type of platform aims to learn about technologies, its use is not limited to small projects. There are large deployments with them that are functional.

8.2. Objectives

The main objective of this project is to analyze the operation of different types of networks formed with ESP8266 devices, according to the technology with which networks are implemented. For this purpose, two protocols and two topologies have been compared and combined, giving rise to four different test scenarios. Some more specific ones can be derived from this objective:

- Study and document the current status of the devices used for low-cost and easy-to-use IoT.
- Study the different technologies that can be used to implement IoT networks with this type of devices.
- Study the different topologies and protocols used in the selected IoT devices.
- To carry out the analysis and comparison of operation between the selected networks.

Once the different types of networks have been analysed and compared, a prototype of a real home automation system using one of the technologies analysed will be implemented. The main objective of the implementation of this prototype is to check that the network

is useful for the intended application, and from it the following specific objectives can be specified:

- Design the complete system: specifications, functionality, components and technologies to be used.
- Implement the complete network, all the nodes and components that form it.
- To carry out the final test and check that the specifications are met.

8.3. Document organization

This document is organized in different chapters, with the following structure:

- **Chapter 1: Introduction.** This chapter presents the project, including motivation, objectives and structure of the project.
- **Chapter 2: State of art.** This second section describes the current state of the art of low cost devices for IoT networks that are also easy to program, presenting the most used ones.
- **Chapter 3: Development environement.** It lists the physical materials and computer platforms and tools used to carry out the project.
- **Chapter 4: Theorical framework.** This chapter develops the theorical framework of the technologies used in the project.
- **Chapter 5: Analysis and results obtained.** This section analyses the different networks and comments on the results obtained.
- **Chapter 6: Real application scenario.** In order to give a realistic approach to the project, a real application has been implemented for a specific scenario of one of the networks analysed.

- **Chapter 7: Conclusions.** Finally, the work done is summarized and the implications are mentioned.

Capítulo 9

Conclusions (English)

9.1. Conclusions and project implications

This project has studied and analyzed some Internet of Things network implementations made with ESP8266 microchips. For this purpose, several test programs have been developed for recollecting data during the operation of each type of network in order to analyze and conclude some characteristics in their behaviour. Once the data of all networks have been obtained and analyzed, a real application has been implemented with one of these technologies.

The developed real scenario has been a domotic house with an alert system to detect anomalies in the behaviour of the house's tenant. The objective of the alerts is to inform the person's relatives or responsible services so the needed help can be received in time if necessary. In order to make the first prototype of this system, it has been necessary to consider what functionalities the system should have, what modules it should include and how to implement it.

The main objective specified at the beginning of this document was the analysis of different types of ESP8266 networks. As it has been shown along this document, and specifically in the Chapter 4, the comparaison between four networks with combined topologies and protocolos has been made. These four analyzed networks have been the mesh topology with MQTT and TCP protocols and the star topology with the same MQTT and TCP protocols.

The results obtained have shown that the star topology with any of the protocols produces worse quality in terms of network reliability than the mesh topology. In this one, the differences between the performance of the protocols are not very noticeable, so it is convenient to select one of them evaluating the needs of the final application. Although this analysis can be extended to many combinations of topologies and protocols, the main objective has been achieved, because the analysed networks are two of the most representative ones. The mesh topology and the MQTT protocol are technologies widely used in IoT systems, while the star topology and the TCP protocol is widely used in the conventional Internet.

Another of the objectives proposed at the beginning was the implementation of a real system with one of the studied technologies. The implementation of this system is described in Chapter 6 of this document, so this objective is also considered to have been achieved. The system has been implemented with mesh topology and MQTT protocol. Analyzing the project needs MQTT was more adjusted to them. MQTT is thought for the communication between machines and also allows to select the quality of the service in the message sending, something convenient for the application. However, the development of this type of system can be made much more complex by adding more sensor modules and new functionalities. As it is a mesh network with modular devices with MQTT protocol, the possibility of implementing and adding new modules to the system is open, and it is not too difficult.

At a personal level, the development of this project has involved the acquisition of new knowledge about Internet of Things networks and the way to implement them. With the analysis of the different types of networks I have learned which parameters are necessary for the comparison and how to obtain them. For the implementation of the real scenario I have learned to face a design from scratch of a real network system, from devising it to implementing it.

9.2. Future work

This project is proposed as the introduction to an analysis of different networks of ESP8266 devices, so the most inmediate future line of work should be the comparison between more types of networks, topologies and protocols.

There are also possible improvements in the analyses done in this work, such as improving the precision of the delay times between the reciving and reception of network messages or including new parameters that allow to determine more network characteristics.

The actual prototype application system also has a lot of things to be improved if a viable market product is to be explored. Some of the things that should be added are more sensor modules and real time analysis of the data obtained to calibrate and custom alert systems inmediatly.

Bibliografía

[1] Arduino. ¿qué es arduino? https://www.arduino.cc/, 2019.

[2] Varios autores Arduino Libraries. Librería de cliente ntp para arduino. https://github.com/arduino-libraries/NTPClient, 2019.

[3] Luis del Valle Hernández. Esp8266 todo lo que necesitas saber del módulo wifi para arduino. https://programarfacil.com/podcast/esp8266-wifi-coste-arduino/#Especificaciones_del_chip_ESP8266, 2017.

[4] Luis del Valle Hernández. Guía de introducción a mqtt con esp8266 y raspberry pi. https://programarfacil.com/esp8266/mqtt-esp8266-raspberry-pi/, 2019.

[5] Redes Cableadas e Inalámbricas. Topología de estrella. https://redesinalambricasycableadas.wordpress.com/redes-cableadas/diferentes-topologias-de-red/topologia-de-estrella/, -.

[6] Varios autores ESP8266 Arduino. Librería wifi para esp8266 para arduino. https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi, 2019.

[7] Alberto A. Del Barrio Jesús Martín Alonso. A distributed hw-sw platform for fireworks. https://dl.acm.org/doi/10.5555/3015574.3015591#pill-authors__contentcon, 2016.

[8] knolleary. Librería mqtt para esp8266 para arduino. https://github.com/knolleary/pubsubclient, 2019.

[9] Jesús Martín; Barrio Alberto A. del; Botella Guillermo Laclaustra, Iván Manuel; Alonso. Sistema domótico distribuido para controlar el riego y el aire acondicionado en el hogar. https://digibug.ugr.es/handle/10481/41915, 2016.

[10] Vicente Lahoz. El protocolo mqtt: impacto en españa. https://www.securityartwork.es/2019/02/01/el-protocolo-mqtt-impacto-en-espana/, 2019.

[11] painlessMesh. Librería painlessmesh para arduino. https://gitlab.com/painlessMesh/painlessMesh, 2019.

[12] PhracturedBlue. Librería esp8266mqttmesh para arduino. https://github.com/PhracturedBlue/ESP8266MQTTMesh, 2019.

[13] Github: PhracturedBlue. Mesh construction and communication. https://github.com/PhracturedBlue/ESP8266MQTTMesh, 2018.

[14] Espressif IOT Team. Esp8266, mesh user guide. Technical report, Espressif Inc, 2016.

[15] Espressif IOT Team. Esp-idf, esp-mesh programming guide. Technical report, Espressif Inc, 2019.

[16] Carlos Villagómez. Protocolo de control de transmisión. https://es.ccm.net/contents/281-protocolo-tcp, 2017.

[17] Michael Yuan. Conociendo mqtt. https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html, 2018.

Apéndice A

Hojas de cálculo

En este Apéndice se recogen los enlaces de las hojas de cálculo de Documentos de Google en los que se han analizado los resultados obtenidos para cada uno de los tipos de redes.

- Topología *mesh* con protocolo TCP: https://docs.google.com/spreadsheets/d/1GTGHCLmihOUcUFvZBw80IUl6En8GAg_FkZmJffLrct0/edit?usp=sharing
- Topología *mesh* con protocolo MQTT: https://docs.google.com/spreadsheets/d/1BK_O8FrxgvoAmy1Q4jwg7rSFHmY_d5vAgZSAYruLjfA/edit?usp=sharing
- Topología estrella con protocolo TCP: https://docs.google.com/spreadsheets/d/1xOBOlgpjme-WH4v38Hlo7ya4LrwROkIl5h2YEhmFiXc/edit?usp=sharing
- Topología estrella con protocolo MQTT: https://docs.google.com/spreadsheets/d/1aBQaoDjppXXXJoFz4EUMcBuSGvNBGLxaDjpPFgJpinM/edit?usp=sharing

Apéndice B

Código de los programas utilizados

Todo el código de los programas implementados para el desarrollo de este trabajo de fin de máster está en el repositorio de Github: https://github.com/aldaMarMu/TFM_code.git.